НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

«До захисту допущено»
Завідувач кафедри
_____ О.І. Ролік
«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6. 050201 «Системна інженерія»

на тему: «Система оптимізації представлення інформаційного поля WEB-сторінки в залежності від потужностей користувача»

Виконав (-ла):
студент (-ка) IV курсу, групи ІА-52
Сокирко Дмитро Борисович _____

Керівник:
Старший викладач Моргаль О. М.
_____

Рецензент:
_____ _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2019 рік

АНОТАЦІЯ

бакалаврської дипломної роботи Сокирка Дмитра Борисовича

на тему: “ Система оптимізації представлення інформаційного поля WEB-сторінки в залежності від потужностей користувача”

Дипломна робота присвячена розгляду додатків для браузерів та їх внутрішніх методів оптимізації, розробці оптимального додатку для браузеру на мовах HTML, JavaScript і CSS. Результатом роботи став додаток, розроблений за допомогою технологій створення додатків, зроблений професійний огляд основних підходів, що використовуються при їх розробці. Були досліджені також такі підходи, як ліниве завантаження, зображення домінуючого кольору, зображення низької якості (LQIP), та технології для отримання доступу до об’єктної моделі веб-сторінки . Було розглянуто основні недоліки та переваги вищезазначених технологій при застосуванні в розробці додатків. Також було розглянуто основні методи оптимізації та розробки, правила та рекомендації, що є оптимальними для побудови веб-сторінок . В результаті була спроектована та побудована система для оптимізації веб-сторінок.

Загальний об’єм роботи 68 сторінки, 31 рисунок, 15 посилань.

Ключові слова: додаток для браузеру, ліниве завантаження, JavaScript, HTML, CSS.

SUMMARY

Bachelor diploma work Dmytro Borysovych Sokyrko

on the theme: "The system of optimizing the presentation of the information field of the WEB-page, depending on the capabilities of the user"

Thesis is devoted to the consideration of applications for browsers and their internal optimization methods, the development of an optimal application for the browser in the languages HTML, JavaScript and CSS. The result of the work was an application developed with the help of application development technologies, a professional review of the main approaches used in their development. The following were also explored: lazy loading, dominant color image, low-quality image (LQIP), and technology for accessing an object model of a web page. The main drawbacks and advantages of the above-mentioned technologies when applied in the development of applications were considered. Also discussed were the main methods of optimization and development, the rules and recommendations that are optimal for the construction of web pages. As a result, a system for optimizing web pages was designed and built.

Total volume of work 68 pages, 31 drawings, 15 references.

Keywords: browser application, lazy boot, JavaScript, HTML, CSS.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

API	Application Programming Interface, програмний інтерфейс
TCP/IP	Transmission Control Protocol/Internet Protocol
HTML	HyperText Markup Language, мова для створення веб-сторінок
JSON	JavaScript Object Notation, текстовий формат обміну даними
HTTP	HyperText Transfer Protocol, протокол передачі даних
БД	База даних
СУБД	Система управління базами даних
LQIP	Low Quality Image Placeholder
DNS	Domain name server
JS	JavaScript, мова програмування
CSS	Cascading Style Sheets
CDN	Content Delivery Network

ВСТУП

Коли постає питання, яким чином пришвидшити завантаження WEB-сторінки, при повільному з'єднанні, в громадських місцях чи в віддалених регіонах, багатьом на думку спадає можливість браузера переходити в «Режит турбо». Проте в цій роботі буде розглянуто зовсім іншу технологію.

Додаток для браузера – це мікропрограма, яка розширює (доповнює) функціонал браузера, вона також здатна через браузер вбудовуватися в хмарний софт, і розширити його функціонал. Їх називають browser extensions (розширення браузера), plug-in (плагін), або add-on (доповнення), і зараз вони підтримуються практично всіма браузерами.

Кожнен з цих додатків має свій власний набір можливостей і кожен різниться в здатності виконувати ті чи інші задачі. Наприклад, «Google Перекладач» встановлений з магазину додатків браузеру, дає можливість перекладу виділеного слова на сторінці, виділеного абзаца та самої сторінки в цілому. Звичайно цих додатків є досить багато та всі вони в загальному створені для спрощення інтерфейсу, що в наслідок тягне за собою збільшення швидкості комфорту при відвідуванні WEB-сторінок, проте для спеціальних функцій для певної WEB-сторінки необхідний свій додаток, який буде виконувати свої функції.

Основною задачею даного дипломного проекту – розробка системи з методом оптимізації відображення WEB-сторінки, розглянення переваг та недоліків, порівняти наявні методи, що створені для оптимізації завантаження WEB-сторінки при повільному з'єднанні.

Основний упор в розробці системи бажано зосередити на оптимізацію зображень, оскільки вони є одним з поширених та найбільшим носієм інформації для користувача. Оскільки розглянути та запам'ятати зображення, яке підкріплено текстом та назвою, буде більш оптимальним для користувача

1. ОСНОВНІ НАПРЯМИ ВИКОРИСТАННЯ ДОДАТКІВ ТА ПЛАГІНІВ. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

Для покращення якості інтернету в віддалених місцях потрібно витрачати багато ресурсів, а також часу. Оскільки кожен провайдер має різне технічне забезпечення, специфічну логіку роботи, райони покриття ,що відрізняються ландшафтом. Для того, щоб задовільнити користувачів інтернетом, який буде достатнім для задоволення мінімальних потреб необхідно виконати заміну обладнання на нове, що не буде раціональним, оскільки через невелику кількість користувачів, для покриття затрат, ціна збільшиться. Тому часто провайдери нехтують даною проблемою та технічне забезпечення залишається на колишньому рівні**.** Тому є додатки та плагіни для браузера, що оптимізують відображення та завантаження веб сторінок в браузері.

Додаток для браузера – це мікропрограма, яка розширює (доповнює) функціонал браузера, вона також здатна через браузер вбудовуватися в хмарний софт, і розширити його функціонал. Їх називають browser extensions (розширення браузера), plug-in (плагін), або add-on (доповнення), і зараз вони підтримуються практично всіма браузерами.

Для створення розширення потрібно знати HTML, CSS, JavaScript, і продумати його функціональне навантаження. Готове розширення надається користувачеві у вигляді архіву, або викачується через офіційний магазин розширень браузера (Chrome, Opera, Firefox, і інші), і просто активується.

Головне призначення розширень зрозуміло вже з самої назви – розширити функціонал браузера. Плагіном можна вбудувати свої елементи в будь-який хмарний сервіс з метою інтеграції, додаткових можливостей для SaaS, створення мікросервіса, або просто як додатковий інструмент.

1.1 Інтеграційні розширення

Хмарний софт для CRMок, програм управління складом і багатьох інших систем по веденню бізнесу стає все більш популярним, а ніякої хмарний провайдер не дає можливості вбудуватися в нього, і підігнати під себе. В такому випадку розширення – класна можливість взяти добре продуманий хмарний сервіс, і вмонтувати в нього одну або кілька своїх кнопок. Їй можна привласнити функцію експорту даних і налагодити процес обміну даними між різними внутрішніми системами. Таким чином можна отримати кастомний функціонал до чужого хмарного софту. Додаткові можливості вашого SaaS в будь-якій вкладці браузера

Це можуть бути фонові сервіси різного характеру: перевірки, підказки, календарі, перекладачі, помічники та інші.

1.2 Мікросервіси

Це мікропрограми, які викликаються «по кнопці» і взаємодіють зі сторінкою. Потрібні вони для сервісів, які має сенс запускати в середовищі іншого сайту додатка. Він здатний зчитувати інформацію з сайту, на якому його викликали, і тим самим істотно спростити користувачеві взаємодію. Це можуть бути будь-які сервіси заміток, скріншотів або відео, вони можуть не просто зберігати дані, а відразу вивантажувати в хмару, або відправляти, наприклад поштою.

1.3 Переваги та недоліки використання додатків

При такому підході рішення проблем можна позначити деякі переваги:

- Швидкість доступу, комфортність і зрозумілість інтерфейсу;

- Кросплатформність – властивість працювати на любій платформі де є браузер;

- Можливість інтегрувати не інтегровне, додати свою функціональність в сторонні продукти в ядро яких доступу нема;

- Можливість об’єднувати свої системи та хмарні сервіси в комплексний кооперативний ландшафт систем;

Звичайно, у такого підходу є свої недоліки:

- По-перше, додатки та скрипти, що встановлюються в браузер, навантажують оперативну пам’ять комп’ютера.

- По-друге**,** необхідність періодичних оновлень додатку під оновлення браузеру чи сервісу;

- По-третє під кожен браузер потрібно писати окрему версію додатку;

1.4 Огляд наявних рішень

1.4.1 Режим турбо

Режим «Турбо» - корисна функція браузерів «Яндекс», Opera, яка дозволяє прискорювати завантаження сторінок сайтів при повільному з'єднанні з інтернетом.

Придумали режим Turbo розробники браузера Opera в 2009 році. Тоді інтернет був у багатьох ще повільним (телефонні модеми) і тарифи припускали оплату за кожен мегабайт прийнятої або відправленої інформації, а режим

дозволяв реально економити. Зараз доступ в мережу у більшості безлімітний, але прискорювати завантаження все також актуально на мобільних з'єднаннях, WiFi в громадських місцях.

Принцип роботи режиму «Турбо» у Opera і «Яндекс Браузер» однаковий. З відключеною опцією користувач завантажує сайт безпосередньо на свій комп'ютер, а при активованому режимі «Турбо» дані спершу викачуються на сервер Opera Software і вже звідти сторінка відкривається у вкладці браузера. На сервері Opera Software мультимедіа - картинки, відео, анімація, - стискаються і при повільному з'єднанні сайти запускаються на комп'ютері користувача швидше - менше обсяг завантажувати інформацію. Якість відео та іншого помітно погіршується, зате подивитися ролик, анімацію або картинку вийде навіть на повільному мобільному інтернеті.

За рахунок того що інтернет браузер підключається не прямого відношення до вебсайту, а через сервера Opera Software, в режимі «Турбо» можна відвідувати сайти, заблоковані або вашим інтернет провайдером. Доступ до заборонених ресурсів блокується на рівні провайдера - поставщики інтернету не дають своїм абонентам заходити на сторінки з певними адресами. У режимі «Турбо» підключення йде відразу до серверів Opera або Google, якщо ви користуєтеся Chrome, тому провайдер не фіксує візит на заборонені сайти і заблокувати не може.

Коли ви заходите з включеним турбо режимом браузера на сайт, який визначає ваш IP адреса, ваше місце розташування або провайдера, наприклад на головну сторінку https://2ip.ru, то можна побачити, що дані визначені невірно. Якщо бути точніше, наш сервіс визначає IP адреса сервера, який забезпечує роботу турбо режиму і на підставі його визначає провайдера і ваше місце розташування.

1.4.2 Плагін низького споживання трафіку «Data saver»

Сервіс «Data Saver» для десктопних версій браузерів скорочує трафік, одержуваний при завантаженні сторінок, за допомогою серверів Google: при включеному розширенні сервери Google стискають дані перед завантаженням веб-сторінок і перевіряють їх на віруси. Для стиснення цей режим працює відмінно - на окремих сайтах «зрізає» до 70% зайвої мультимедіа - рекламних банерів, анімації та ін., - а ось в якості засобу для заходу на заблоковані сайти не сильно підходить.

Рисунок 1.1 - Плагін Data saver

1.4.3 Браузер Vivaldi

Браузер Vivaldi побудований на базі Chromium і підтримує установку розширень для Chrome. Реалізувати режим стиснення трафіку в цьому браузері

можна за допомогою того ж розширення «Економія трафіку». Vivaldi примітний вбудованими функціями і ефектами для веб-сторінок, в числі яких:

- блокувальник зображень;
- блокувальник банерів і режим читання.

Режим читання на борту Vivaldi можна використовувати як інструмент економії трафіку, оскільки при його активації для кожного окремого сайту статейні матеріали такого сайту будуть відкриватися постійно в режимі читання - очищеному від веб-елементів книжковому форматі. Установити блокування Adobe Flash можна в налаштуваннях Vivaldi, в розділі «Веб-сторінки».

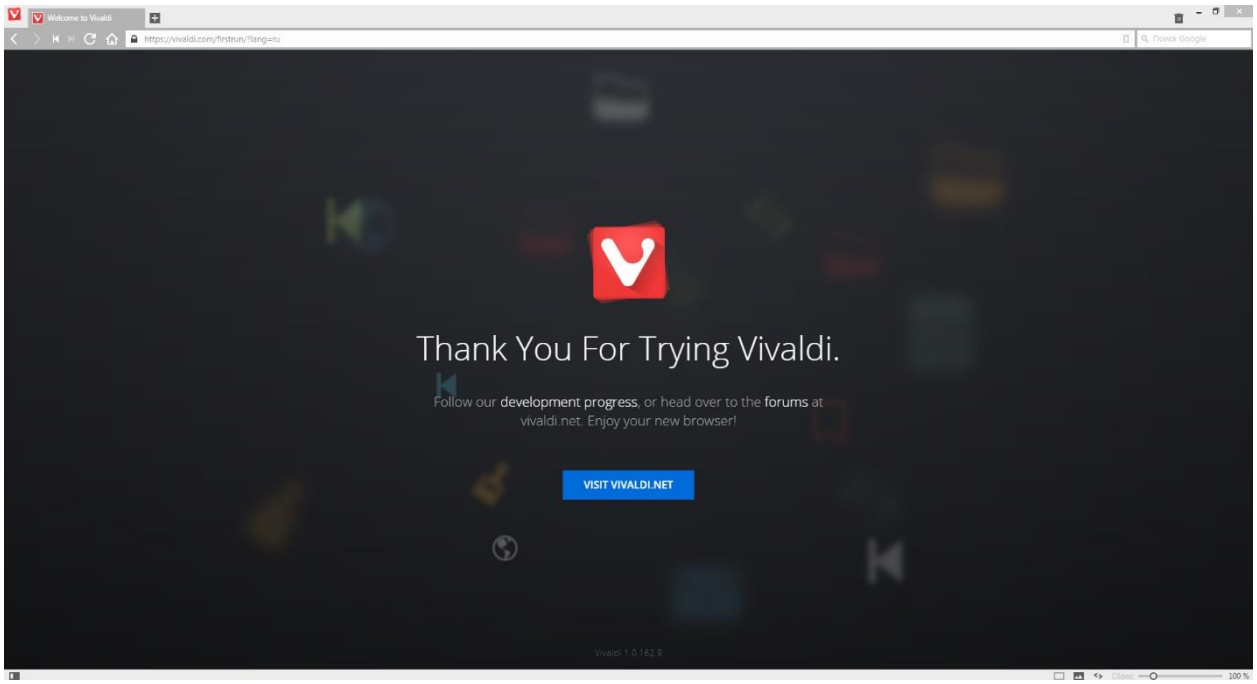

Рисунок 1.2 - Браузер Vivaldi

ВИСНОВКИ

Розглянувши доступні рішення систем оптимізації відображення інформаційного поля веб-сторінки можна зробити висновок, що вони мають свої переваги та недоліки.

Дані методи направлення на спрощення відображення та завантаження інформації, рішення з використанням серверів, для стиснення та перенаправлення інформації дає високий відсоток ефективності, але може викликати деякі незручності у зв’язку зі зміною регіону, де той чи інший ресурс може бути заблоковано.

Браузер Vivaldi від Vivaldi corporation перейняв багато функцій з браузеру Opera, а також були розроблені свої, такі як :блокувальник баннерів і режим читання. Даний режим читання доволі цікава функція, яка працює не при ввімкненні на веб-сторінці, а при включенні і працює на всих сторінках що були, чи будуть відкриті. Але, для того щоб в громадському місці зайти на кілька ресурсів встановлення нового браузеру не буде оптимальним.

Рішення від Google Chrome з плагіном “Data saver” є оптимальним для використання, оскільки плагін є універсальним рішенням для браузерів, але даний плагін був закритий компанією Chrome та перенесений на платформу Android і названий “Lite mode”.

2 ДОСЛІДЖЕННЯ МЕТОДУ ФОРМУВАННЯ WEB-СТОРІНКИ

Всі знають, що повільний сайт - це погано. Через гальмуючого сайту виникають серйозні проблеми при вирішенні повсякденних завдань. Іноді це просто дратує. Часто гальмування сайту - це і поломка, відмова в обслуговуванні - люди не чекають завантаження і йдуть. Це актуально для випадків радикального гальмування сайту, наприклад, коли початок відтворення сторінки починається через 8-10 секунд після кліка.

Навіть при відносно благополучною ситуацію з сайтом (при швидкій завантаженні на дротовому інтернеті і сучасному комп'ютері), затримки в завантаженні можуть призводити до втрат аудиторії і зниження конверсії. Наприклад, компанія Amazon проводила експеримент, в якому з'ясувала, що кожні 100 мс (0,1 с) затримки призводять до зниження продажів на 1%.

Також важливим є питання швидкості сайту - технічна. Як правило, повільні сайти споживають підвищений обсяг ресурсів хостингу, який призводить до додаткових витрат. Гальма серверної частини знижують можливості без проблемно переживати піки навантаження на сайт.

Тому швидкістю сайту потрібно займатися як з технічної, так і з економічної точок зору.

2.1 Швидкість сайту: основні компоненти

Швидкість сайту стосується двох сторін: клієнтської і серверної. На сьогоднішній день кожна з цих частин рівнозначна для кінцевого результату. Але кожна зі своїми особливостями.

Щоб зрозуміти з чого формується час завантаження сторінки сайту, розберемо цей процес на етапи. В результаті ми зможемо зрозуміти, де знаходяться можливості серверної і клієнтської оптимізації.

Повний процес завантаження сайту (перші відвідини) виглядає наступним чином:

1. DNS-запит на ім'я сайту.
2. Підключення до сервера по IP (TCP-підключення).
3. Встановлення захищеного з'єднання при використанні HTTPS (TLS-підключення).
4. Запит HTML-сторінки по URL і очікування сервера. (HTTP-запит)
5. Завантаження HTML.
6. Розбір HTML-документа на стороні браузера, побудова черги запитів в ресурсам документа.
7. Завантаження та парсинг CSS-стилів.
8. Завантаження і виконання JS-коду.
9. Початок рендеринга сторінки, виконання JS-коду.
10. Завантаження веб-шрифтів.
11. Завантаження зображень і інших елементів.
12. Закінчення рендеринга сторінки, виконання відкладеного JS-коду.

У цьому процесі деякі позиції відбуваються паралельно, деякі можуть мінятися місцями, але суть залишається тією ж.

Серверна оптимізація займається етапами з першого по четвертий включно. Етапи з 5 по 12 - це клієнтська оптимізація. Час, витрачений на кожен з цих етапів, індивідуально для кожного сайту, тому необхідно отримувати метрики сайту і виявляти основне джерело проблем. І тут ми переходимо до питання про те, як ці метрики отримати і інтерпретувати.

2.1.1 DNS-запит

DNS - комп'ютерна розподілена система для отримання інформації про домени. Найчастіше використовується для отримання IP-адреси по імені хоста

(комп'ютера або пристрою), отримання інформації про маршрутизації пошти, які обслуговують вузлах для протоколів в домені (SRV-запис).

Розподілена база даних DNS підтримується за допомогою ієрархії DNS-серверів, що взаємодіють за певним протоколом.

Основою DNS є уявлення про ієрархічну структуру доменного імені та зонах. Кожен сервер, який відповідає за ім'я, може делегувати відповідальність за подальшу частину домену іншого сервера (з адміністративної точки зору - інший організації або людині), що дозволяє покласти відповідальність за актуальність інформації на сервери різних організацій (людей), що відповідають тільки за «свою» частину доменного імені.

Починаючи з 2010 року в систему DNS впроваджуються засоби перевірки цілісності переданих даних, звані DNS Security Extensions (DNSSEC). Передані дані не шифруються, але їх достовірність перевіряється криптографічними способами. Впроваджуваний стандарт DANE забезпечує передачу засобами DNS достовірної криптографічного інформації (сертифікатів), які використовуються для встановлення безпечних і захищених з'єднань транспортного і прикладного рівнів.

DNS має наступні характеристики:

- Розподіленість адміністрування. Відповідальність за різні частини ієрархічної структури несуть різні люди або організації.
- Розподіленість зберігання інформації. Кожен вузол мережі в обов'язковому порядку повинен зберігати тільки ті дані, які входять в його зону відповідальності, і (можливо) адреси кореневих DNS-серверів.
- Кешування інформації. Вузол може зберігати деяку кількість даних не зі своєї зони відповідальності для зменшення навантаження на мережу.

- Ієрархічна структура, в якій всі вузли об'єднані в дерево, і кожен вузол може або самостійно визначати роботу нижчестоящих вузлів, або делегувати (передавати) їх іншим вузлам.
- Резервування. За зберігання та обслуговування своїх вузлів (зон) відповідають (зазвичай) декілька серверів, розділені як фізично, так і логічно, що забезпечує збереження даних і продовження роботи навіть у разі збою одного з вузлів.

DNS важлива для роботи Інтернету, так як для з'єднання з вузлом необхідна інформація про його IP-адресу, а для людей простіше запам'ятовувати буквені (зазвичай осмислені) адреси, ніж послідовність цифр IP-адреси. У деяких випадках це дозволяє використовувати віртуальні сервери, наприклад, HTTP-сервери, розрізняючи їх по імені запиту. Спочатку перетворення між доменними і IP-адресами проводилося з використанням спеціального текстового файлу hosts, який складався централізовано і автоматично розсилався на кожну з машин в своїй локальній мережі. З ростом Мережі виникла необхідність в ефективному, автоматизованому механізмі, яким і стала DNS.

У всіх DNS повідомлень одинаковий формат:

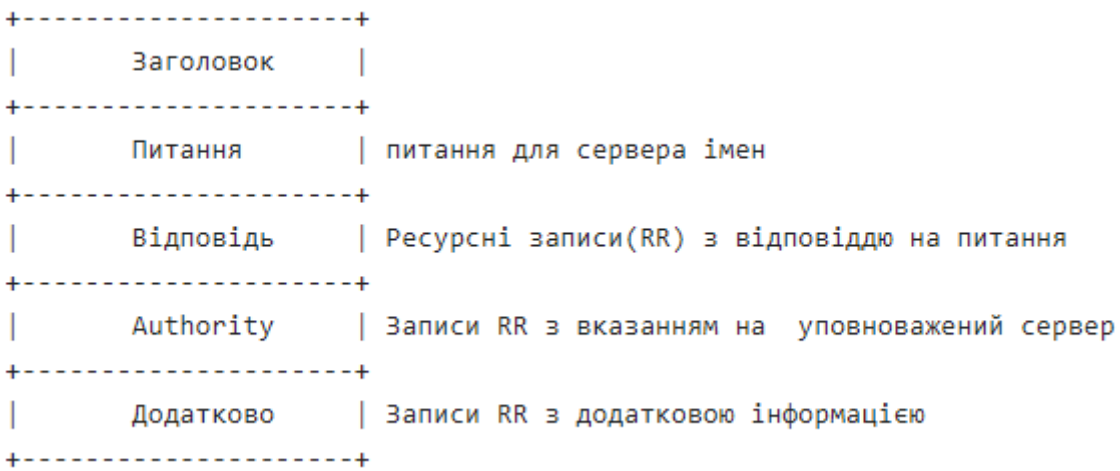

Рисунок 2.2 - Формат DNS повідомлень

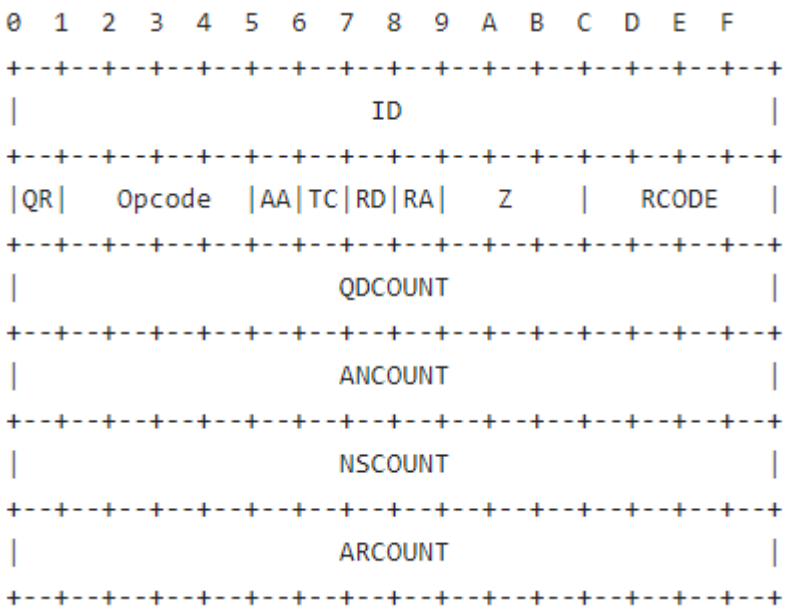

Рисунок 2.3 - Формат заголовка DNS повідомлення

На рисунку 2.1.1.3 кожна комірка представляє єдиний біт. У кожному рядку 16 колонок, що становить два байти даних. Діаграма поділена на рядки для простоти сприйняття, але реальне повідомлення являє собою безперервний ряд байтів.

Оскільки у запитів і відповідей однаковий формат заголовка, деякі поля не мають відношення до запиту і будуть встановлені в значення 0.

Значущими є дані поля:

- ID: Довільний 16-бітний ідентифікатор запиту. Такий же ID використовується у відповіді на запит, так що можна встановити відповідність між ними.
- QR: однобітний прапор для вказівки, є повідомлення запитом (0) або відповіддю (1). Оскільки запит відправляється, то встановлюється 0.

- Opcode: Чотирьохбітне поле, яке визначає тип запиту. Відправляється стандартний запит, так що вказано 0. Інші варіанти:
  - 0: Стандартний запит
  - 1: Інверсний запит
  - 2: Запит статусу сервера
  - 3-15: Зарезервовані для майбутнього використання
- TC: однобітний прапор, який вказує на обрізане повідомлення. В даному випадку коротке повідомлення, його не потрібно обрізати, вказується 0.
- RD: однобітний прапор, який вказує на бажану рекурсію. Якщо DNS-сервер, якому відправляється питання, не знає відповіді на нього, він може рекурсивно опитати інші DNS-сервери. Треба активувати рекурсію, так що вказується 1.
- QDCOUNT: 16-бітове беззнаковое ціле, що визначає число записів в секції питання. Відправляється 1 питання.

Секція питання має наступний формат:

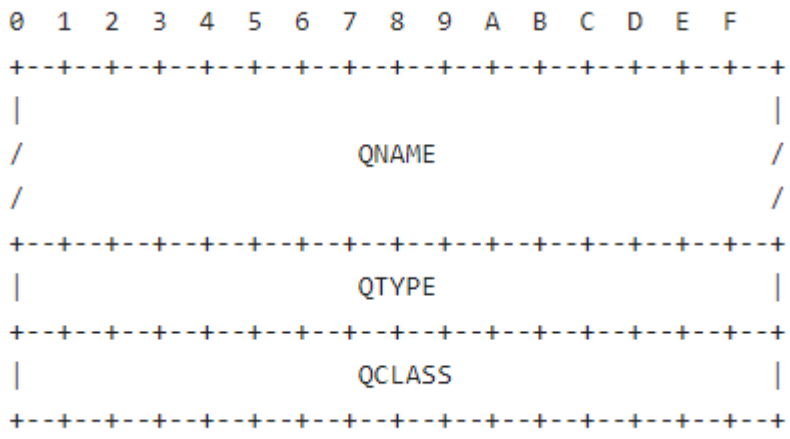

Рисунок 2.4 - Формат секції питання

- QNAME: Ця секція містить URL, для якого ми хочемо знайти IP-адресу. Вона закодована як серія написів (labels). Кожна напис відповідає секції URL. Так, в адресі example.com дві секції: example і com.

  Для складання написи потрібно закодувати кожну секцію URL, отримавши ряд байтів. Надпис - це ряд байтів, перед якими стоїть байт без знакового цілого, що позначає кількість байт в секції. Для кодування нашого URL можна просто вказати ASCII-код кожного символу.

  Секція QNAME завершується нульовим байтом (0).
- QTYPE: Тип запису DNS, яку шукають.
- QCLASS: Клас, який шукають.

2.2 Вимірювання швидкості сайту

Головне питання: що потрібно вимірювати? Існує безліч метрик по швидкості сайтів, але основних не так багато.

По-перше, це час до першого байта (TTFB - time to first byte) - це час від початку процесу завантаження до отримання першої порції даних від сервера. Це основна метрика для серверної оптимізації.

По-друге, це початок рендерингу сторінки (start render, first paint). Метрика показує час до закінчення періоду «білого екрану» в браузері, коли починається формування сторінки.

По-третє, це завантаження основних елементів сторінки (load time). Сюди входить завантаження і інтерпретація всіх ресурсів для роботи зі сторінкою, після цієї позначки індикатор завантаження сторінки перестає крутитися.

По-четверте, це повне завантаження сторінки: час до закінчення основної діяльності браузера, завантажені всі основні і відкладені ресурси.

Ці основні метрики вимірюються в секундах. Також корисно мати оцінку обсягу трафіку для третьої і четвертої метрики. Трафік потрібно знати для оцінки впливу швидкості з'єднання на час завантаження.

Також потрібно зрозуміти, чим тестувати швидкість. Існує безліч сервісів і засобів для оцінки метрик швидкості завантаження сайтів, кожен з яких краще для свого завдання.

Один з найпотужніших інструментів - панель розробника в браузері. Найбільш розвинена функціональність у панелі в браузерах. На вкладці Network можна отримати метрики за часом завантаження всіх елементів, включаючи сам HTML-документ. При наведенні на елемент можна дізнатися, скільки часу витрачено на кожен етап отримання ресурсу. Для оцінки повної картини процесу завантаження сторінки можна скористатися вкладкою Performance, яка дає повну деталізацію аж до часу декодування картинок.

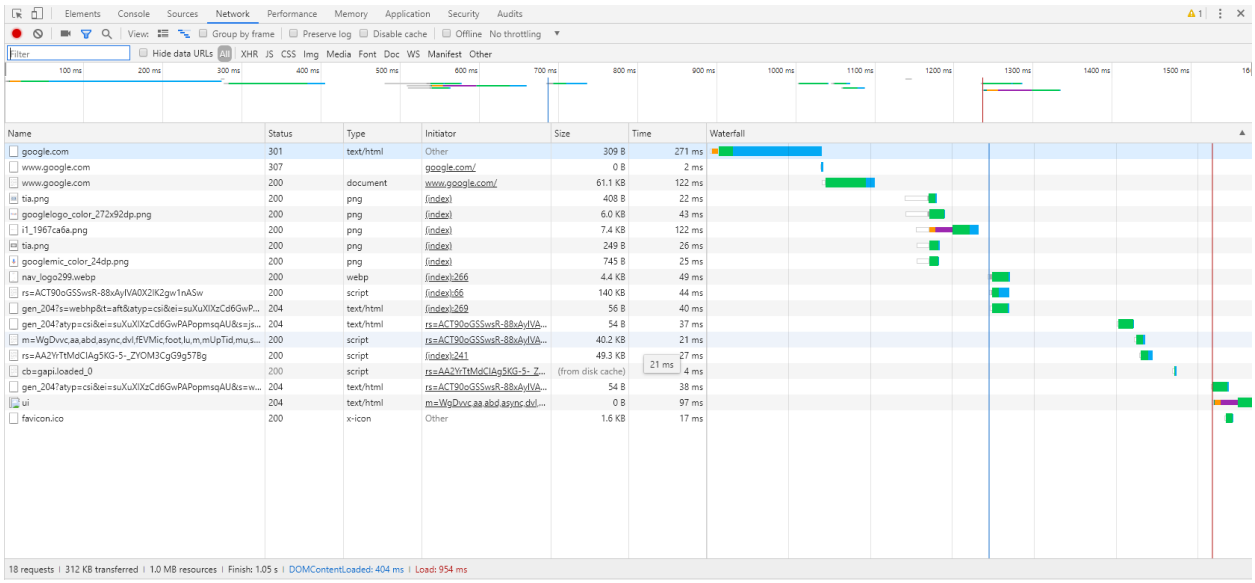

Рисунок 2.5 – Вкладка Network на панелі розробника в браузері Google Chrome

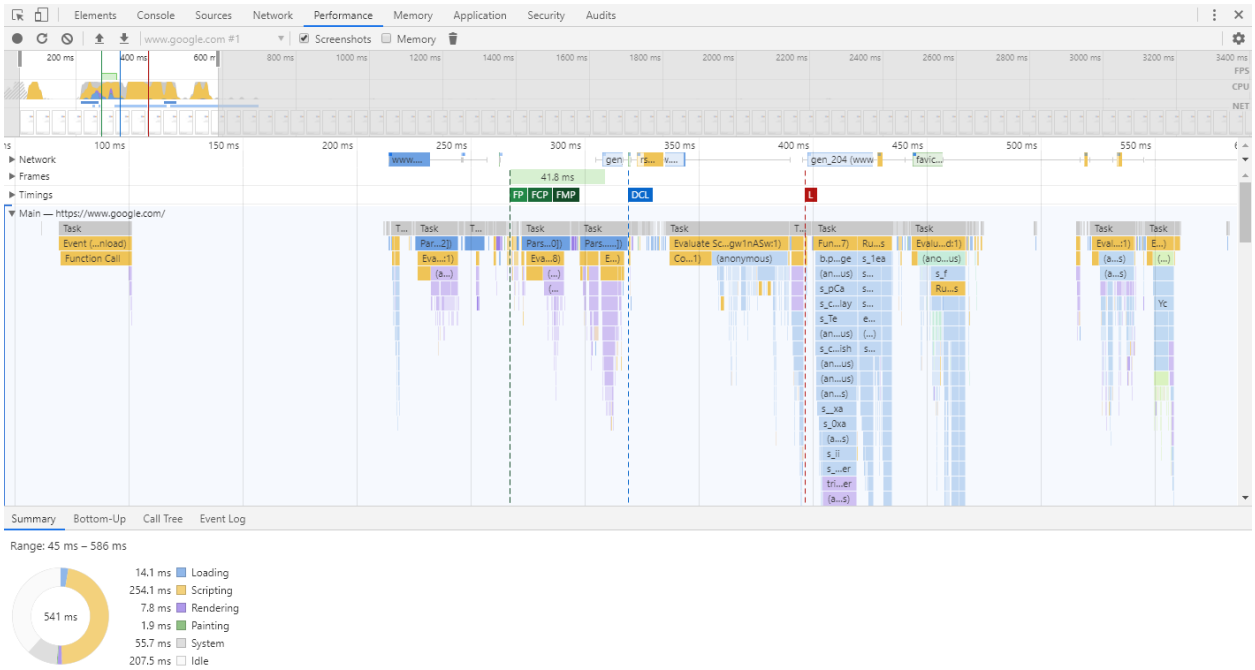

Рисунок 2.6 – Вкладка Perfomance на панелі розробника в браузері Google Chrome

Якщо потрібно оцінити швидкість сайту без повної деталізації, корисно запустити аудит сайту (вкладка Audits), він буде проведений з використанням плагіна Lighthouse. У звіті ми отримуємо оцінку швидкості для мобільних пристроїв (як інтегральну в балах, так по нашим основним метрик) і кілька інших звітів.

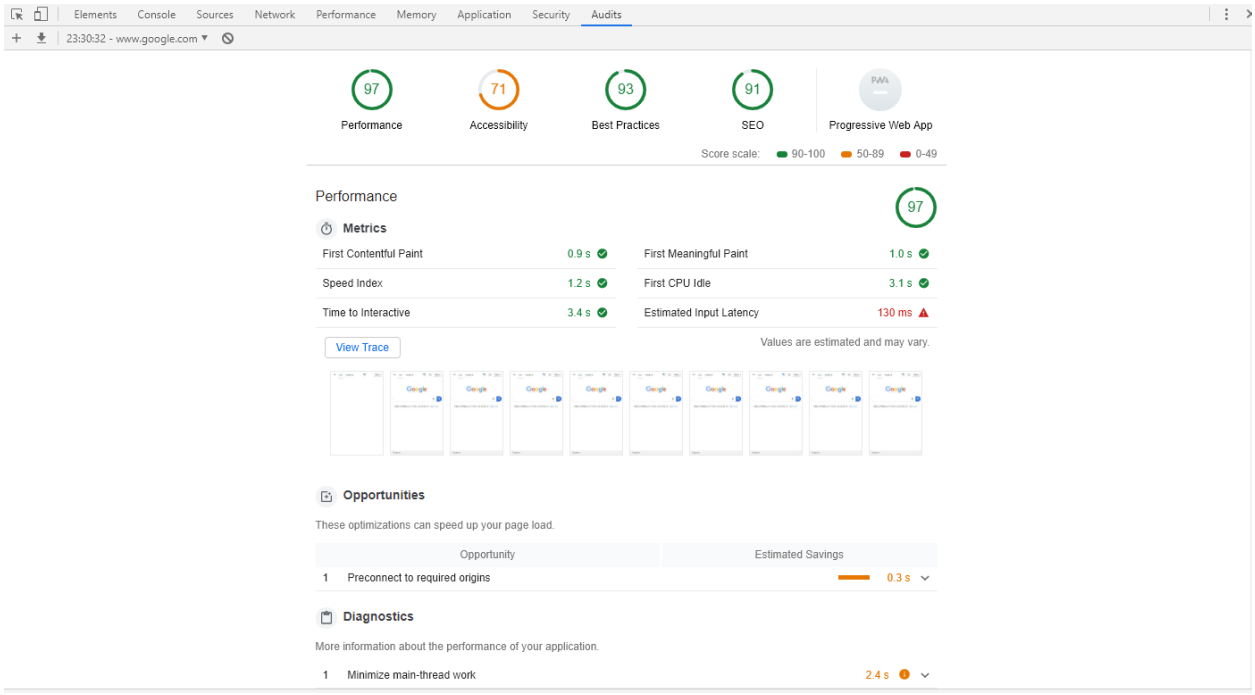

Рисунок 2.7 – Вкладка Audit на панелі розробника в браузері Google Chrome

Серед веб-сервісів професійним стандартом стала система WebPagetest. Це сервіс завантажує сайт в реальних браузерах з заданим типом з'єднання і формує докладний звіт по всіх етапах і метрик.

Performance Results (Median Run)

	Load Time	First Byte	Start Render	Speed Index	Last Painted Hero	First Interactive (beta)	Document Complete			Fully Loaded			
							Time	Requests	Bytes In	Time	Requests	Bytes In	Cost
First View (Run 3)	1.692s	0.511s	0.800s	0.801s	0.900s	> 0.745s	1.692s	15	390 KB	3.653s	19	391 KB	$----

Рисунок 2.8 – Результати тестів сайту google.com на ресурсі WebPagetest

Для швидкої оцінки клієнтської оптимізації можна скористатися сервісом Google PageSpeed Insights, але потрібно пам'ятати, що він не включає в себе більшість найважливіших метрик за часом завантаження. Нарешті, корисно

аналізувати час завантаження сайту у реальних користувачів. Для цього є спеціальні звіти в системах веб-аналітики Яндекс.Метрика і Google Analytics.

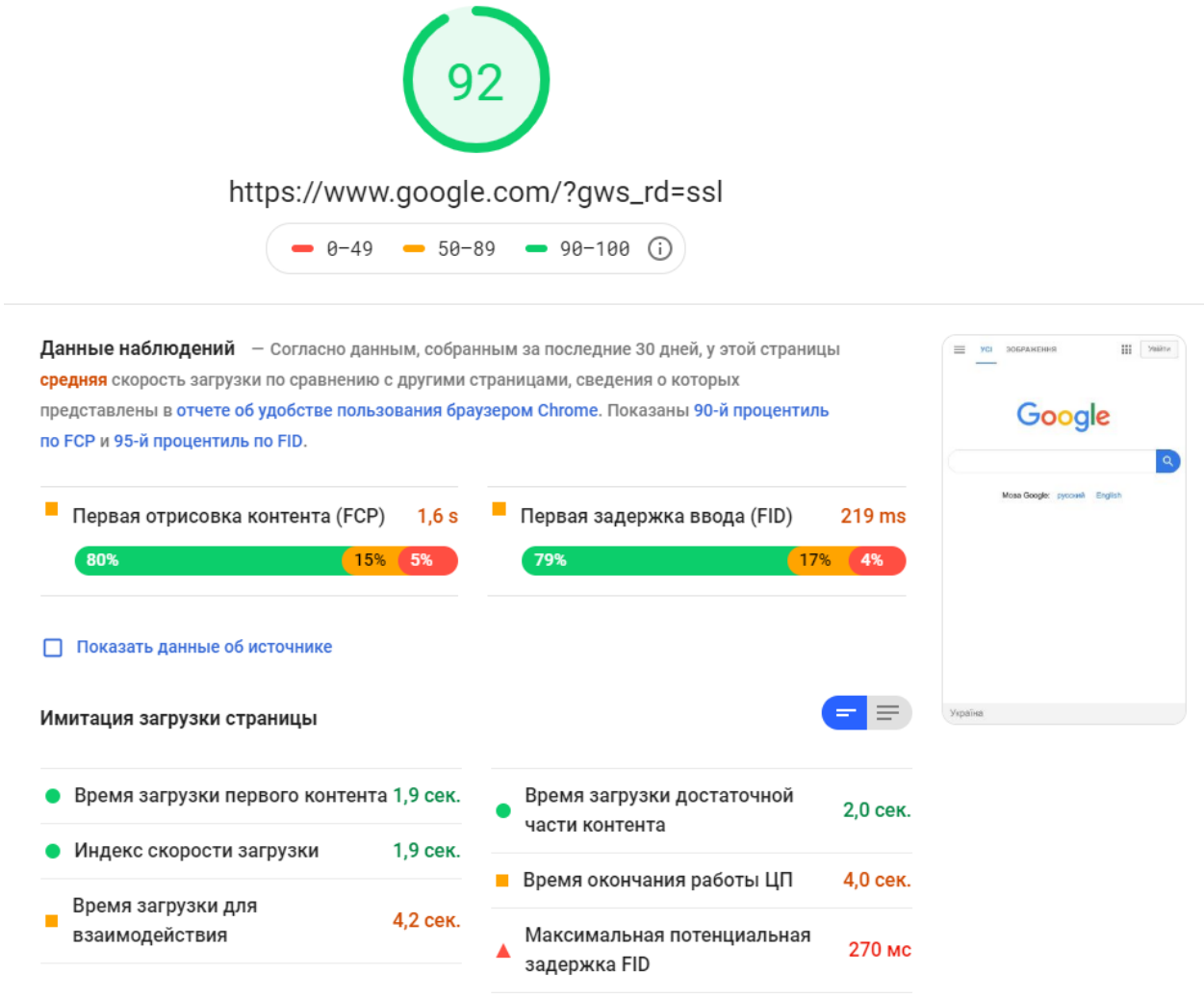

Рисунок 2.9 – Результати тестів сайту google.com на ресурсі pagetest

Орієнтири для часу завантаження сайту такі: початок рендеринга близько 1 секунди, завантаження сторінки в межах 3-5 секунд. У таких рамках користувачі не будуть скаржитися на швидкість сайту і час завантаження не обмежуватиме ефективність сайту. Ці цифри повинні досягатися саме у реальних користувачів, навіть в складних умовах мобільного підключення і застарілих пристроїв.

2.3 Серверна оптимізація

Перейдемо до самого прискоренню сайту. Оптимізація серверної частини - найбільш зрозуміла і очевидна міра для розробників сайту. По-перше, серверна частина досить легко моніториться і контролюється на стороні системних адміністраторів. По-друге, при серйозних проблемах з часом відповіді сервера уповільнення помітно для всіх, незалежно від швидкості підключення або пристрою.

При тому що причини гальмування серверної частини можуть бути найрізноманітніші, є типові місця, на які потрібно подивитися.

2.3.1 Хостинг (серверні ресурси)

Це причина гальмування номер один для невеликих сайтів. Для поточного навантаження сайту просто не вистачає ресурсів хостингу (зазвичай, це CPU і швидкість дискової системи). Якщо можна швидко збільшити ці ресурси, варто спробувати. У деяких випадках проблема буде вирішена. Якщо вартість додаткових ресурсів стає вище, ніж вартість робіт по оптимізації, потрібно переходити до наступних методів.

2.3.2 СУБД (сервер бази даних)

Тут ми вже переходимо до вирішення джерела проблеми: низькій швидкості роботи програмного коду. Часто велика частина часу веб-додатки витрачається на запити до БД. Це логічно, тому що завдання веб-додатки зводиться до збору даних і перетворення їх за певним шаблоном.

Рішення проблеми повільних відповідей від БД зазвичай розділяється на два етапи: тюнінг СУБД і оптимізація запитів і схеми даних. Тюнінг СУБД (наприклад, MySQL) може дати прискорення в кілька разів, в разі, якщо настройка

раніше взагалі не проводилася. Тонкий тюнінг може дати ефект в межах десятка відсотків.

Оптимізація запитів і схеми даних це радикальний спосіб прискорення. За рахунок такої оптимізації можна отримувати прискорення на кілька порядків. Якщо зміна структури БД може відбуватися без вторгнення в програмний код сайту, то оптимізація запитів таке втручання потребують.

Для виявлення повільних запитів потрібно зібрати статистику по навантаженню на БД за досить тривалий проміжок часу. Далі проводиться аналіз логу і виявлення кандидатів на оптимізацію.

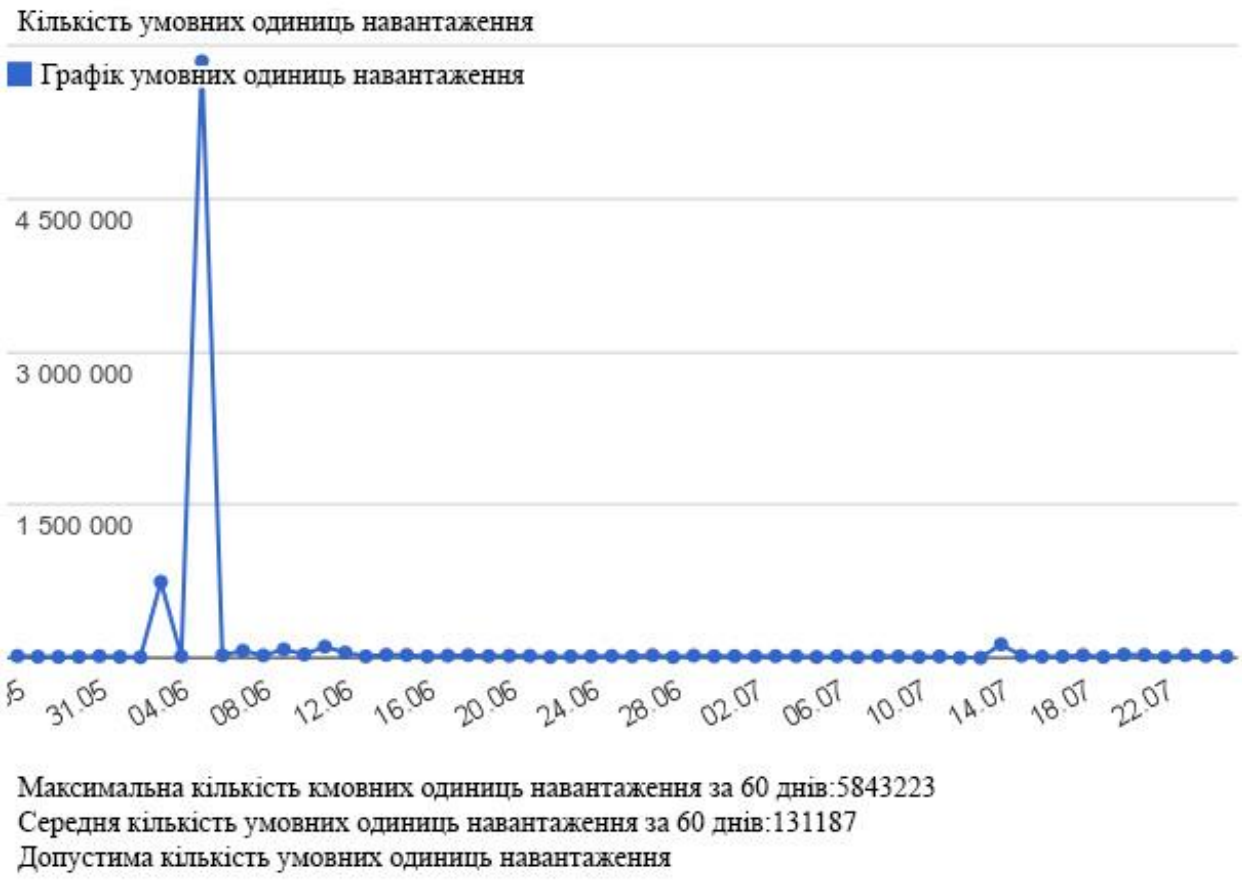

Рисунок 2.10 – Приклад графіку навантаження СУБД в різні дні

2.3.3 Кешування

Найпотужнішим і універсальним засобом збільшення серверної швидкості традиційно є кешування. Тут ми говоримо саме про серверному кешуванні, а не про кешуючих заголовках. Якщо обчислення результату (збірка сторінки, блоку) вимагає значних ресурсів, покладемо результат в кеш і будемо періодично його оновлювати. Ідея проста і складна водночас: системи кешування вбудовані в мови програмування, системи управління сайтами і веб-сервери.

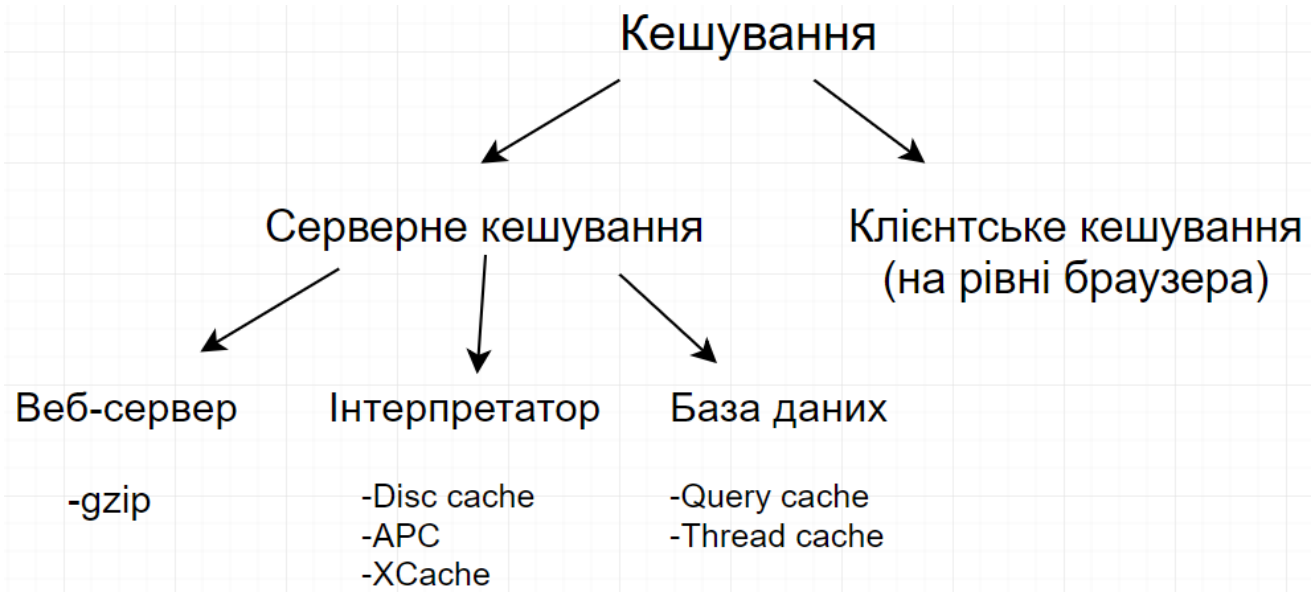

Рисунок 2.11 – Загальна модель кешування з поділом на серверне та клієнтське кешування

Як правило, кешування сторінок дозволяє скоротити час віддачі сторінки до десятків мілісекунд. Природно, що в цьому випадку сервер легко переживає піки відвідуваності. Проблеми тут дві: не все можна закешовану і кеш потрібно грамотно інвалідувати(скидати). Якщо проблеми вирішуються, кешування можна рекомендувати як ефективний засіб серверного прискорення.

2.3.4 Оптимізація TCP, TLS, HTTP / 2

У цій частині ми об'єднали тонкі мережеві оптимізації, які дають серверне прискорення. Ефект тут не такий масштабний, як в інших методах, але досягається виключно налаштуванням, тобто безкоштовний.

Transmission Control Protocol (TCP, протокол управління передачею) - один з основних протоколів передачі даних інтернету, призначений для управління передачею даних.

Механізм TCP надає потік даних з попередньою установкою з'єднання, здійснює повторний запит даних в разі втрати даних і усуває дублювання при отриманні двох копій одного пакета, гарантуючи тим самим, на відміну від UDP, цілісність переданих даних і повідомлення відправника про результати передачі.

Реалізації TCP зазвичай вбудовані в ядра ОС. Існують реалізації TCP, що працюють в просторі користувача.

Коли здійснюється передача від комп'ютера до комп'ютера через Інтернет, TCP працює на верхньому рівні між двома кінцевими системами, наприклад, браузером і веб-сервером. TCP здійснює надійну передачу потоку байтів від одного процесу до іншого. TCP реалізує управління потоком, управління перевантаженням, рукостискання, надійну передачу.

TLS - криптографічні протоколи, що забезпечують захищену передачу даних між вузлами в мережі Інтернет . TLS і SSL використовують асиметричне шифрування для аутентифікації, симетричне шифрування для конфіденційності та коди автентичності повідомлень для збереження цілісності повідомлень.

TLS дає можливість клієнт-серверних додатків здійснювати зв'язок в мережі таким чином, що не можна проводити прослуховування пакетів і здійснити несанкціонований доступ.

Так як більшість протоколів зв'язку може бути використано як з, так і без TLS (або SSL), під час активного з'єднання необхідно явно вказати серверу, чи

хоче клієнт встановлювати TLS. Це може бути досягнуто або за допомогою використання уніфікованого номера порту, по якому з'єднання завжди встановлюється з використанням TLS, або з використанням довільного порту та спеціальної команди сервера з боку клієнта на перемикання з'єднання на TLS з використанням спеціальних механізмів протоколу (як, наприклад, STARTTLS для протоколів електронної пошти). Як тільки клієнт і сервер домовилися про використання TLS, їм необхідно встановити захищене з'єднання. Це робиться за допомогою процедури підтвердження зв'язку. Під час цього процесу клієнт і сервер приймають угоду щодо різних параметрів, необхідних для установки безпечного з'єднання.

Основні кроки процедури створення захищеного сеансу зв'язку:

- клієнт підключається до сервера, що підтримує TLS, і запитує захищене з'єднання;
- клієнт надає список підтримуваних алгоритмів шифрування і хеш-функцій;
- сервер вибирає зі списку, наданого клієнтом, найбільш надійні алгоритми серед тих, які підтримуються сервером, і повідомляє про свій вибір клієнта;
- сервер відправляє клієнту цифровий сертифікат для власної аутентифікації. Зазвичай цифровий сертифікат містить ім'я сервера, ім'я засвідчувального центру сертифікації і відкритий ключ сервера;
- клієнт, до початку передачі даних, перевіряє валідність (автентичність) отриманого серверного сертифіката щодо наявних у клієнта кореневих сертифікатів засвідчувальних центрів (центрів сертифікації). Клієнт також може перевірити, чи не відкликаний чи серверний сертифікат, зв'язавшись з сервісом довіреної засвідчує,;
- для шифрування сесії використовується сеансовий ключ. Отримання загального секретного сеансового ключа клієнтом і сервером

проводиться по протоколу Діффі-Хеллмана. Існує історичний метод передачі згенерованого клієнтом секрету на сервер за допомогою шифрування асиметричною криптосистемою RSA (використовується ключ з сертифікату сервера). Даний метод не рекомендований, але іноді продовжує зустрічатися на практиці.

На цьому закінчується процедура підтвердження зв'язку. Між клієнтом і сервером встановлено безпечне з'єднання, дані, що передаються по ньому, шифруються і розшифровуються за використанням симетричною криптосистеми до тих пір, поки з'єднання не буде завершено.

HTTP / 2 - друга велика версія мережевого протоколу HTTP, яка використовується для доступу до World Wide Web.

Цілі:

- Додати механізми узгодження протоколу, клієнт і сервер можуть використовувати HTTP 1.1, 2.0 або, гіпотетично, інші, що не HTTP-протоколи.
- Підтримати сумісність з багатьма концепціями HTTP 1.1, наприклад по набору методів доступу (GET, PUT, POST і т. П.), Статусним кодами, формату URI, великій кількості заголовків
- Зменшення затримок доступу для прискорення завантаження сторінок, зокрема шляхом:
    - Стиснення даних в заголовках HTTP
    - Використання push-технологій на серверній стороні
    - конвейеризации запитів
    - Усунення проблеми блокування «head-of-line» протоколів HTTP 1.0 / 1.1
    - Мультиплексування безлічі запитів в одному з'єднанні TCP
- Збереження сумісності з широко впровадженими застосуваннями HTTP, в тому числі з веб-браузерами (повноцінними і мобільними),

API, використовуваними в Інтернеті, веб-серверами, проксі-серверами, зворотними проксі-серверами, мережами доставки контенту

При виникненні проблем на деяких з вищевказаних кроків підтвердження зв'язку може завершитися з помилкою, а безпечне з'єднання не буде встановлено.

Даний протокол широко використовується в додатках, що працюють з мережею Інтернет, таких як веб-браузери, робота з електронною поштою, обмін миттєвими повідомленнями і IP-телефонія (VoIP).

Тюнінг TCP сьогодні потрібно для великих проектів і серверів з підключенням від 10G, основне, що потрібно пам'ятати: мережева підсистема регулярно оновлюється з виходом нових ядер Linux, тому варто оновлюватися. Правильне налаштування TLS (HTTPS) дозволяє отримати високий рівень безпеки та максимально скоротити час встановлення захищеного з'єднання. Хороші рекомендації випущені компанією Mozilla.

Нова версія HTTP протоколу - HTTP / 2 покликана прискорити завантаження сайтів. Цей протокол з'явився недавно і зараз активно використовується (близько 20% частки серед веб-сайтів). Загалом, в HTTP / 2 дійсно закладені механізми прискорення, основний - зниження впливу мережевих затримок на час завантаження сторінки (request multiplexing). Але прискорення за рахунок HTTP / 2 далеко не завжди успішно, тому не варто сподіватися на цей протокол.

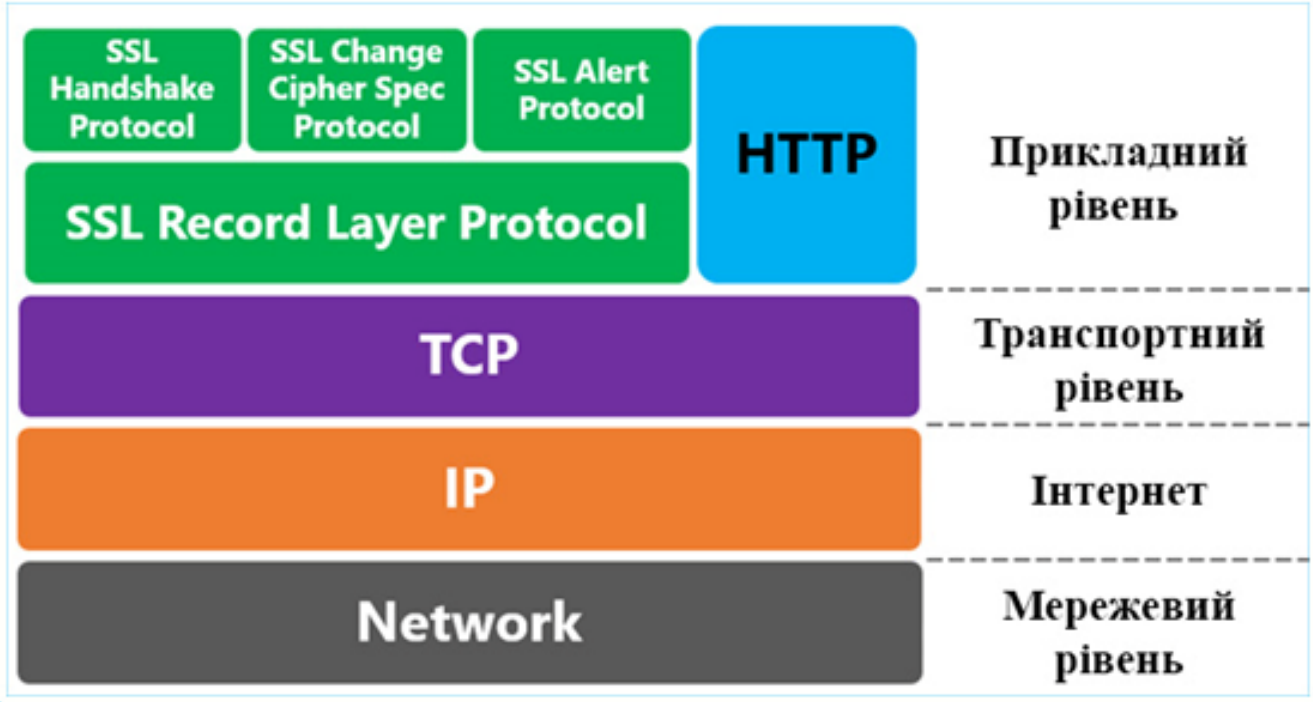

Рисунок 2.12 - TCP HTTP

2.4 Клієнтська оптимізація

На відміну від серверної оптимізації, клієнтська спрямована на все, що відбувається в браузері користувача. Через це ускладнюється контроль (різні пристрої і браузери) і виникає безліч різних напрямків оптимізації. Ми подивимося на найдієвіші і універсальні методи, які можна використовувати практично в будь-якому проекті.

2.4.1 Оптимізація критичного шляху: CSS, JS

Критичний шлях рендеринга (critical rendering path) - набір ресурсів для початку відтворення сторінки в браузері. Як правило, в цей список входять сам HTML-документ, CSS-стилі, веб-шрифти і JS-код.

Перше завдання як оптимізаторів швидкості скоротити цей шлях як за часом (з урахуванням затримок мережі), так і по трафіку (щоб врахувати повільні з'єднання).

Найпростіший шлях визначити критичний шлях: запустити аудит в браузері (в панелі розробника), використовуючи плагін Lighthouse, що визначає його склад і час завантаження з урахуванням повільного підключення.

Основна техніка в скороченні критичного шляху: прибираємо все, що не потрібно або можна відкласти. Наприклад, більшість JS-коду можна відкласти до завантаження сторінки. Для цього розміщуємо виклик JS-ресурсу в кінці HTML-документа або використовуємо атрибут async.

Для відкладеної завантаження CSS можна скористатися динамічним підключенням стилів через JS (дочекавшись події domContentLoaded).

2.4.2 Оптимізація веб-шрифтів

Підключення веб-шрифтів сьогодні стало практично стандартом в дизайні. На жаль, вони негативно впливають на швидкість відтворення сторінок. Веб-шрифти - це додаткові ресурси, які потрібно отримати до початку відтворення тексту.

Ситуація погіршується тим, що часто покажчики на файли шрифтів закопані в CSS-файлі, який також приходить не миттєво. Багато розробники люблять користуватися публічними сервісами веб-шрифтів (наприклад, Google Fonts), що викликає ще більші затримки (додаткові підключення, CSS-файл).

Правила оптимізації полягають в скороченні обсягу трафіку веб-шрифтів і отриманні їх якомога швидше.

Щоб скоротити трафік потрібно використовувати сучасні формати: WOFF2 для сучасних браузерів, WOFF для сумісності. Крім того, потрібно включати

тільки ті набори символів, які використовуються на сайті (наприклад, латиниця і кирилиця).

Web Open Font Format (WOFF) — шрифтовий формат для використання у веб-сторінках. Формат який було розроблено в 2009 році і перебуває у процесі стандартизації як рекомендація спеціалізованої групи Web Fonts Working Group

Вплинути на швидке відображення веб-шрифтів можна за допомогою нових специфікацій «preliad» і CSS-властивості «font-display». Preload дозволить якомога раніше вказати браузеру про необхідність завантаження файлу шрифту, а font-display дає гнучкі можливості по управлінню поведінкою браузера в разі затримки файлу (почекати, выдобразити запасний, не чекати шрифт більше трьох секунд).

Сьогодні в Інтернеті використовуються чотири формату контейнерів шрифтів: EOT, TTF, WOFF і WOFF2. На жаль, незважаючи на можливість вибору, не існує єдиного формату, який працює у всіх браузерах. Наприклад, EOT доступний тільки в IE, TTF підтримується в цьому браузері тільки частково. WOFF поширений найширше, однак його не можна використовувати в деяких старих браузерах, а над підтримкою WOFF 2.0 працюють в даний час.

Отже, єдиного формату для всіх браузерів не існує, тому необхідно використовувати кілька форматів, щоб сторінка виглядала однаково у всіх користувачів:

- Використовувати WOFF 2.0 в тих браузерах, які його підтримують.
- Додавати WOFF для більшості браузерів.
- Застосовувати TTF в застарілих браузерах Android (для версій до 4.4).
- Додавати EOT для підтримки застарілих версій IE (до IE9).

2.4.2.1 Зменшення розміру шрифту за допомогою стиснення

Шрифт - це набір гліфів, кожен з яких складається з контурів букв. Звичайно, все гліфи різні, але, тим не менш, вони містять багато схожою інформації, яку можна стиснути за допомогою GZIP або іншого сумісного компресора.

Формати EOT і TTF не стискуються за замовчуванням. Потрібно переконатися,що на серверах налаштоване стиснення GZIP при передачі ресурсів в цих форматах.

До формату WOFF застосовується вбудоване стиснення. Потрібно переконатися,що компресор використовує оптимальний варіант відповідних налаштувань.

WOFF2 використовує для користувача алгоритми попередньої обробки і стиснення для зменшення розміру файлу на 30% в порівнянні з іншими форматами.

Варто відзначити, що деякі формати шрифтів містять додаткові метадані, наприклад інформацію про гінтинг і кернінг:

Гі́нтинг — прив’язка форми векторної літери шрифту до сітки, щоб уникнути імовірних деформацій. Полягає у позначенні ключових моментів шрифтової форми так званими гінтами, тобто спеціальними інструкціями. Особливо потрібен для малих розмірів літер чи при низькій роздільності відтворюючого пристрою.

Кернінг - процес зміни розмірів міжбуквенних пропусків (інтервалів) між сусідніми буквами для поліпшення зовнішнього вигляду і легкості читання тексту. Цей параметр відповідає за індивідуальну роботу з кожною буквою і підбір її місцеположення залежно від вибраного шрифту, малюнка самої букви та її сусідніх букв, смислового навантаження слова

Вони не потрібні не для всіх платформ, тому ми можемо стиснути файл ще більше. Дізнайтеся про відповідні функції вашого компресора шрифтів і переконайтеся, що у вас є відповідні ПО для тестування ресурсів і надання їх браузерам. Наприклад, Google Fonts підтримує більше 30 оптимізованих варіантів кожного шрифту і автоматично визначає і застосовує оптимальний варіант для кожної платформи і браузера.

2.4.3 Оптимізація зображень

Зображення складають більшість ваги сучасного сайту. Звичайно, картинки це не такі критичні ресурси для сторінки, як CSS- і JS-код. Але для безлічі сайтів зображення складають важливу частину контенту: згадаємо будь-яку картку товару в інтернет-магазині.

Основна методика при оптимізації зображень: скорочення їх розміру. Для цього потрібно використовувати правильний формат і інструменти стиснення:

PNG для картинок з прозорістю і текстом;

JPEG для фото і складних зображень;

SVG для векторної графіки.

На додаток до цих форматів існують нові: наприклад, WebP від Google. WebP - формат стиснення зображень з втратами і без втрат якості, запропонований компанією Google Inc. в 2010 році. Заснований на алгоритмі стиснення нерухомих зображень (ключових кадрів) з видеокодека VP8. Використовує контейнер RIFF. Для роботи з даним форматом існує відкрите програмне забезпечення, зокрема бібліотека libvpx і конвертер webpconv.

Цей формат може покрити область використання PNG і JPEG - підтримує стиснення з втратами і без втрат, прозорість і навіть анімацію. Для його

використання досить створити копії картинок в WebP і віддавати їх браузерам, які їх підтримують.

Для PNG існує безліч утиліт по оптимізації, які можна використовувати для скорочення розміру, наприклад, OptiPNG, PNGout, ect і інші. Також внутрішню оптимізацію стиснення даних можна проводити за допомогою zopfliPNG. Основна ідея такого софта в підборі оптимальних параметрів компресії, видаленні зайвих даних з файлу. Тут потрібно бути обережним: деякі утиліти мають режим з втратою якості, що може не підходити вам (якщо ви очікуєте на виході точно таку ж картинку).Оптимізація JPEG також поділяється на два типи: з втратами і без втрат. В цілому можна порекомендувати пакет Mozilla JPEG, який спеціально розроблений для кращого стиснення в цьому форматі. Для оптимізації без втрат можна використовувати jpegtran, з втратами – cjpeg.

Таблиця 2.1 – Порівняння розмірів зображень з різними форматами

	Розмір зображення 1	Розмір зображення 2	Розмір зображення 3
WebP	1,8 МБ	293 КБ	1,6 МБ
JPG	3,9 МБ	1,3 МБ	3,5 МБ
GIF	6,3 МБ	3,9 МБ	6,7 МБ
PNG	13,2 МБ	5 МБ	12,5 МБ

2.4.4 Кешуючі заголовки

Це найбільш проста методика клієнтської оптимізації. Її сенс у кешуванні браузером рідко змінюючих ресурсів: картинок, CSS і JS-файлів, шрифтів, іноді навіть самого HTML-документа. В результаті кожен ресурс запитується з сервера тільки один раз.

Якщо ви використовуєте Nginx, досить додати директиву:

add_header Cache-Control "max-age = 31536000, immutable";

З цього моменту браузер має право кешувати ресурси на термін до року (що практично назавжди). Новий параметр «immutable» говорить про те, що ресурс не планується змінювати.

Звичайно, виникає питання: а що робити, якщо нам потрібно змінити закешовану ресурс? Відповідь проста: змінити його адресу, URL. Наприклад, можна додати версію в ім'я файлу. Для HTML-документів такий метод також застосуємо, але, як правило, використовується більш короткий термін кешування (наприклад, одна хвилина чи година).

2.4.5 Стиснення даних

Обов'язкова практика це стиснення будь-яких текстових даних при передачі від сервера браузеру. Більшість веб-серверів мають реалізацію gzip-стиснення відповідей.

Однак, простої активації стиснення недостатньо.

По-перше, ступінь стиснення регулюється і має бути близькою до максимальної.

По-друге, можна використовувати статичне стиснення, тобто попередньо стиснути файли і покласти на диск. Тоді веб-сервер буде шукати стислу версію і відразу її видавати.

В-третє, можна використовувати більш ефективні алгоритми стиснення: zopfli (сумісний з gzip) і brotli (новий алгоритм стиснення). Brotli буде працювати тільки з HTTPS. Так як ці алгоритми (особливо zopfli) затратні при стисненні, обов'язково використовуємо їх в статичному варіанті.

Для максимального ефекту стиснення на файли попередньо застосовується процес мініфікації: очищення від непотрібних перекладів рядків, прогалин та

інших непотрібних символів. Цей процес специфічний для кожного формату. Також варто подбати про стиснення інших текстових даних на сайті.

2.5 Використання CDN

Застосування CDN (content delivery network) для прискорення сайтів дуже розрекламована міра, що має багато маркетингової лушпиння навколо суті технології.

Спочатку CDN були розроблені для розвантаження інтернет-каналів мовлення медіасайтов. Наприклад, при перегляді відео в прямому ефірі кілька тисяч глядачів створюють дуже велике навантаження на пропускну здатність сервера. Крім того, забезпечити безперебійне якість зв'язку при значній відстані клієнта і сервера вкрай складно (через затримки і нестабільності мережі).

Вирішення цієї проблеми було у створенні CDN, тобто розподіленої мережі, до якої підключалися клієнти (наприклад, глядачі), а хости цієї мережі вже до сервера (origin). При цьому кількість підключень до сервера скорочувалася до одного (декількох), а кількість підключень до CDN могло досягати мільйонів за рахунок кешування контенту мережею.

Сьогодні більшість CDN позиціонують себе як засіб прискорення сайтів, в першу чергу за рахунок скорочення відстані від контенту до клієнта (відвідувача сайту).

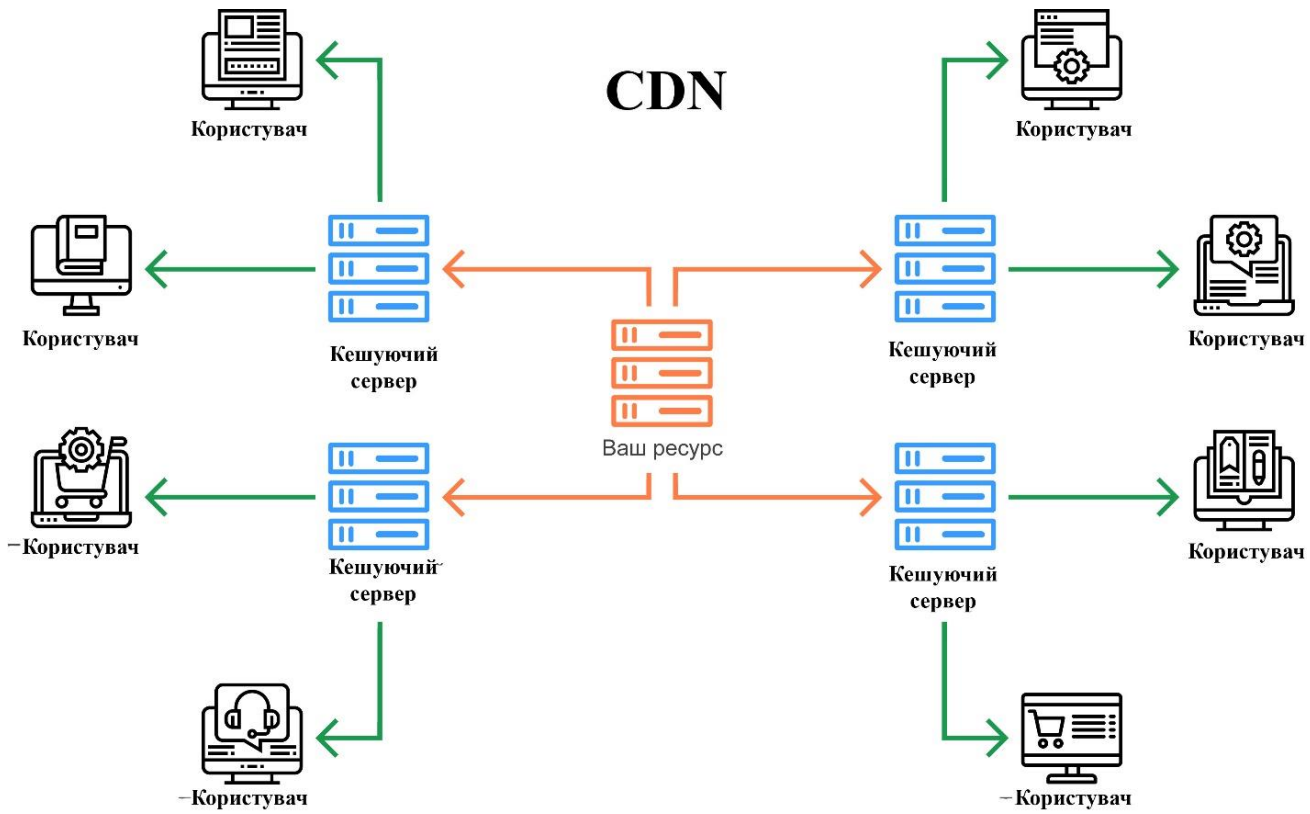

Рисунок 2.13 – схема роботи CDN

2.5.1 Прискорення сайту за допомогою CDN

Так, дійсно користувач підключається, як правило, до ближнього (за часом доступу) сервера мережі та отримує швидкий процес встановлення TCP і TLS-з'єднання. Далі, якщо контент знаходиться на сервері CDN, користувач може швидко його отримати. Таким чином, знижується навантаження на наш власний сервер.

По-друге, CDN може не просто роздавати контент без змін, а оптимізувати його на своєму боці і віддавати в більш компактному вигляді: стискати зображення, застосовувати компресію до тесту і т. Д. За рахунок таких оптимізацій можна отримати скорочення часу завантаження.

2.5.2 Недоліки використання CDN

Недоліки, як зазвичай, продовження гідності: об'єкт може бути не в кеші вузла CDN. Наприклад, він ще не запитували чи його не можна кешувати (HTML-документ). В цьому випадку ми отримуємо додаткові затримки між вузлом CDN і нашим сервером.

Незважаючи на те що CDN покликані прискорювати доступ до сайту, можливі ситуації, коли мережевий маршрут буде менш оптимальним, ніж без CDN.

Врешті решт , мережі доставки контенту це дуже складні системи, в яких також як і всюди можливі збої, нестабільність та інші проблеми. Використовуючи CDN, ми додаємо ще один рівень складності.

ВИСНОВКИ

В даному розділі було розглянуто методи за допомогою яких досягається оптимізація на етапі розробки, а також методи та застосунки, що використовуються під час введення в експлуатацію та підтримки, веб-сервера та веб-сторінки в цілому.

Також розглянуто: переваги та недоліки використання тих чи інших методів, оптимальність використання форматів зображень та порівняння по критерію розмір-якість, завантаження та використання шрифтів на сайті, скриптів, які використовуються для впровадження певних методів та функцій.

3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ - ДОДАТКУ ДЛЯ БРАУЗЕРА

Плагін - це доповнення (розширення можливостей) для будь-якої програми на вашому комп'ютері або движка сайту в інтернеті. Розробникам дуже важко буває передбачити всі побажання користувачів, тому вони дають можливість стороннім розробникам задовольняти ці побажання за допомогою написання плагінів (від англійського plugin).

Крім того, якщо всі можливі речі передбачити в одному додатку, то воно стане дуже важким і неповоротким, а завдяки плагінам цього не відбувається, бо кожен користувач отримує базовий функціонал, а все інше можна отримати, скачавши або встановивши потрібне розширення.

Розглянемо дану концепцію на прикладі браузерів. Розширення для браузера - це комп'ютерна програма, що працює в зв'язці з оглядачем і наділяє його новими можливостями. Термінологія при цьому в різних програмах може відрізнятися. Вони можуть називатися доповненнями (add-on), плагінами (plug-in), розширеннями (extension), але суті це не міняє.

Розробка деяких розширень не слідує за розробкою браузерів і може виникнути така ситуація, що після оновлення браузера якісь з них працювати перестануть. Щось вдіяти з цим складно, але популярні розробники додатків тримають планку і намагаються викочувати оновлені версії своїх плагінів своєчасно.

Сучасна концепція розвитку програмного забезпечення часто має на увазі включення в основну програму тільки базових функцій, а додаткова функціональність реалізуються за допомогою доповнень. Браузери є типовими представниками такої ідеології.

Звичайно, потрібно розуміти, що базова функціональність будь-якої програми річ непостійна. Програми постійно розвиваються, обростаючи новими

можливостями і відкидаючи непотрібні. Те, що спочатку з'явилося, як доповнення згодом може увійти в базову функціональність програми. Варто відзначити, що найчастіше розробниками додатків є абсолютно сторонні люди або організації ніяк не пов'язані з розробниками основної програми.

Родоначальником використання сторонніх програмних блоків для розширення функціоналу браузера був Mozilla firefox. Власне, сам браузер став набрав популярності через те, що вперше була впроваджена ідея розширення функціоналу до нескінченності за допомогою плагінів і встиг набрати на цьому популярність серед користувачів, поки основні конкуренти не почали впроваджувати ідею та повторяти його досвід.

Оскільки браузери зараз є дуже важливими програмами для людини, і є в будь-якому комп'ютері, а часом і кілька відразу, а їх творці надають необхідну інформацію для написання плагінів бажанням будь-кого, їх створено величезну кількість на будь-який випадок.

3.1 Відкладене завантаження зображень

Зображення складають цілих 65% всього веб-контенту і час завантаження сайту легко може стати проблемою. Навіть при правильній оптимізації зображення можуть бути важкими. Це може негативно позначитися на часі, який користувач буде витрачати на очікування, щоб отримати доступ до будь-чого. Швидше за все, відвідувач не дочекається і підуть кудись в інше місце, якщо не буде придумане рішення для завантаження зображень таким чином, щоб не заважати сприйняттю швидкості.

Зображення важливі для кожного сайту і додатку. Неможливо уявити собі сайт без рекламного банера, зображення продукту або логотипу.

Але зображення часто багато важать, і це в першу чергу впливає на розмір сторінки. Згідно з даними сайту HTTP Archive, середня вага сторінки на

комп'ютері складає 1511 КБ. Зображення займають майже 650 КБ, що приблизно 45% від загального числа.

Тепер важливо зробити швидке завантаження сторінки із зображеннями. Розглянемо метод, який допоможе скоротити час завантаження сторінки і зменшити її розмір, не жертвуючи картинками.

Це метод, який використовують при розробці сайтів і додатків, він дозволяє відкласти завантаження зображень на сторінці на більш пізній момент часу. Зображення завантажуються не відразу при відкритті сторінки, а тільки коли з'являється необхідність. Це допомагає поліпшити продуктивність і економно використовувати ресурси пристрою.

Метод відкладеної завантаження можна застосувати практично до всього на сторінці. Наприклад, якщо в односторінковому додатку JavaScript-файл не потрібен користувачу до певного моменту, то краще взагалі не завантажувати його. Якщо зображення не потрібно відразу, як тільки користувач відкрив сторінку, можна завантажити його пізніше, коли воно дійсно знадобиться.

Особливо корисно це буде для користувачів, які тут же закривають сторінку або взаємодіють тільки з її верхньою частиною.

3.2 Інструменти

Основна ідея проста - відкласти завантаження всього, що не потрібно користувачеві прямо зараз. До будь-якого зображення, яке користувач не бачить спочатку, можна застосувати цей метод.

Коли користувач прокручує сторінку вниз, плейсхолдери зображень переходять в область перегляду (видима частина сторінки). І зображення завантажуються, коли стають видимими.

За допомогою розширення Lighthouse для браузера Google Chrome можна дізнатися, які саме зображення підходять для відкладеної завантаження і скільки трафіку можна заощадити. У розширенні є розділ, присвячений закадровим зображень.

3.3 Способи реалізації

Зображення на сторінці можна завантажувати двома способами - за допомогою тегу <img> або за допомогою CSS-властивості «background», яке дозволяє встановити одночасно кілька характеристик фону. Спочатку буде розглянуто більш поширений тег <img>, а потім буде перехід до фонових зображень CSS.

3.4 Тег для відображення зображення на WEB-сторінці

Відкладене завантаження зображень можна розділити на два етапи.

Крок перший - запобігти початкову завантаження зображення. Для зображень, завантажених за допомогою тега <img />, браузер використовує атрибут тега «src» для запуску завантаження зображення. Не має значення, перше це або тисячне зображення в HTML і закадровий воно. Якщо браузер отримає атрибут «src», це викличе завантаження зображення.

Щоб завантажити зображення через відкладену завантаження, потрібно помістити URL-адресу зображення в атрибут «src». Припустимо, вказуємо URL-адресу зображення в атрибуті «data-src» тега «image». Тепер, коли «src» порожній, браузер не почне завантажувати зображення.

```
<img data-src="https://ik.imagekit.io/demo/default-image.jpg" />
```

Рисунок 3.1 – URL-адреса поміщена в «data-src»

Другий крок - потрібно задати установку браузеру, коли завантажувати зображення. Для цього треба встановити, що як тільки зображення (тобто його плейсхолдер) потрапляє в вікно перегляду, починається завантаження. Щоб перевірити, чи дісталося зображення у вікно перегляду, існує два способи.

3.5 Завантаження зображень за допомогою подій JavaScript

У цьому методі використовуємо відстеження подій прокрутки (scroll), зміни розміру (resize), зміни орієнтації (orientationChange) в браузері.

Коли відбувається одне з цих подій, знаходяться всі зображення на сторінці, які ще не завантажені. Перевіряється, які з них наразі перебувають у вікні перегляду. Це можна визначити за допомогою властивостей «offset top», «scroll top» і «window height».

Якщо зображення увійшло в вікно перегляду, береться URL з атрибуту «data-src» і поміщається в атрибут «src». Це запускає завантаження зображення. Також видаляєтсья клас, який визначає зображення, які будуть завантажуватися пізніше. Після завантаження всіх зображень видаляються інструменти для відстеження подій.

Коли відбувається прокручування, подія прокрутки швидко спрацьовує кілька разів. Для підвищення продуктивності додається невеличкий тайм-аут, який регулює відкладене виконання функції завантаження.

Перші три зображення повинні завантажуватися заздалегідь. Тому перші три URL треба вказати безпосередньо в атрибуті «src» замість атрибута «data-src».

Це необхідно для оптимального користувацького досвіду. Оскільки ці зображення знаходяться у верхній частині сторінки, їх слід зробити видимими якомога швидше. Користувач не повинен чекати події або виконання JavaScript, щоб завантажити цю частину сторінки.

3.5.1 Завантаження забражень за допомогою Intersection Observer API

Intersection Observer API - відносно новий API в браузерах. Він визначає, коли елемент входить у вікно перегляду, і починає діяти. У попередньому методі доводилося пов'язувати події, враховувати продуктивність і підраховувати час появи елемента у вікні перегляду.

Intersection Observer API робить процес простіше, допомагає уникнути обчислень і забезпечує хорошу продуктивність.

Як тільки API виявляє, що елемент увійшов в вікно перегляду, використовуючи властивість «isIntersecting», вибирається URL з атрибуту «data-src» і переміщається в атрибут «src», щоб запустити відкладену завантаження. Як тільки це буде зроблено, видаляємо клас, який визначає зображення, які будуть завантажуватися пізніше, з зображення, а також звідти видаляється обсервер.

Якщо порівняти час завантаження зображення двох методів - з відстеженням подій і Intersection Observer API, - то можна виявити, що за допомогою Intersection Observer API завантаження зображення запускається набагато швидше, і сайт вже не виглядає «млявим» при скролінгу.

У метод, який використовує відстеження подій, довелося включити тайм-аут, щоб зробити його працездатним, що негативно впливає на роботу користувача, оскільки зображення завантажуються з невеликою затримкою.

Однак, як і все нове, підтримка Intersection Observer API може бути недоступною в браузерах. Таким чином, доводиться повертатися до методу відстеження подій в браузерах, де Intersection Observer API не підтримується.

3.6 Відкладене завантаження фонових зображень CSS

Після тегів <img /> фонові зображення є найбільш поширеним способом завантаження зображень для сторінок. Для тегів <img /> в браузері простий підхід - якщо URL-адреса зображення доступна, то можна його завантажити.

З фоновими зображеннями CSS не все так просто. Щоб завантажити фонові зображення CSS, браузер повинен створити дерево DOM (об'єктна модель документа), а також дерево CSSOM (об'єктна модель CSS), щоб вирішити, чи застосовується стиль CSS до вузла DOM в поточному документі.

Якщо правило CSS, що визначає фонове зображення, не застосовується до елементу в документі, то браузер не завантажує фонове зображення. Якщо застосовується - завантажує.

Спочатку це може здатися складним, але такий же принцип лежить в основі техніки відкладеної завантаження фонових зображень. Так можна отримати можливість обманути браузер, не застосовуючи властивість CSS «background-image» до елементу, поки цей елемент не потрапить у вікно перегляду.

Тут слід зазначити, що код JavaScript для відкладеного завантаження можна використовувати той самий що й в завантаженні зображень за допомогою Intersection Observer API. Можена використовувати Intersection Observer API, повертаючись потім до відстеження подій. Хитрість полягає в CSS.

Елемент з ідентифікатором «bg-image» має заданий властивість «background-image» в CSS. Однак коли клас ,який визначає зображення, які будуть завантажуватися пізніше, додається до цього елементу, в CSS ми перевизначаємо властивість «background-image» і міняємо його на значення «none».

Так як за правилами комбінація «bg-image» з класом, що визначає завантаження, має більш високу перевагу в CSS, ніж просто «bg-image», браузер застосовує властивість «background-image: none» до елементу спочатку.

Коли прокручується сторінка вниз, the Intersection Observer (або відстеження подій) визначає, що зображення знаходиться у вікні перегляду, і видаляє клас, що визначає зображення. Це змінює застосовуваний зараз CSS і застосовує властивість «background-image» до елементу, що почав завантаження фонового зображення.

3.6.1 Ліниве завантаження

При ледачому завантаженні зображення на сайті завантажуються асинхронно, тобто після повного завантаження видимої частини сторінки або взагалі за умовою тільки при появі у view-порте. Це означає, що якщо користувач не долиста сторінку до кінця, зображення в нижній частині взагалі не буде завантажено.

Даний підхід використовується на деяких сайтах, але особливо помітний на сайтах з великою кількістю зображень. Якщо зайти на онлайн-майданчик фотографій у високій роздільній здатності і буде відразу зрозуміло, як сайт завантажує обмежену кількість зображень. У міру прокручування вниз буде видно, як плейсхолдери швидко замінюються на реальні зображення. Наприклад, можна розглянути сайт Unsplash.com: прокрутка цієї частини сторінки в видиму частину екрану викликає заміну плейсхолдера фотографією високої роздільної здатності:

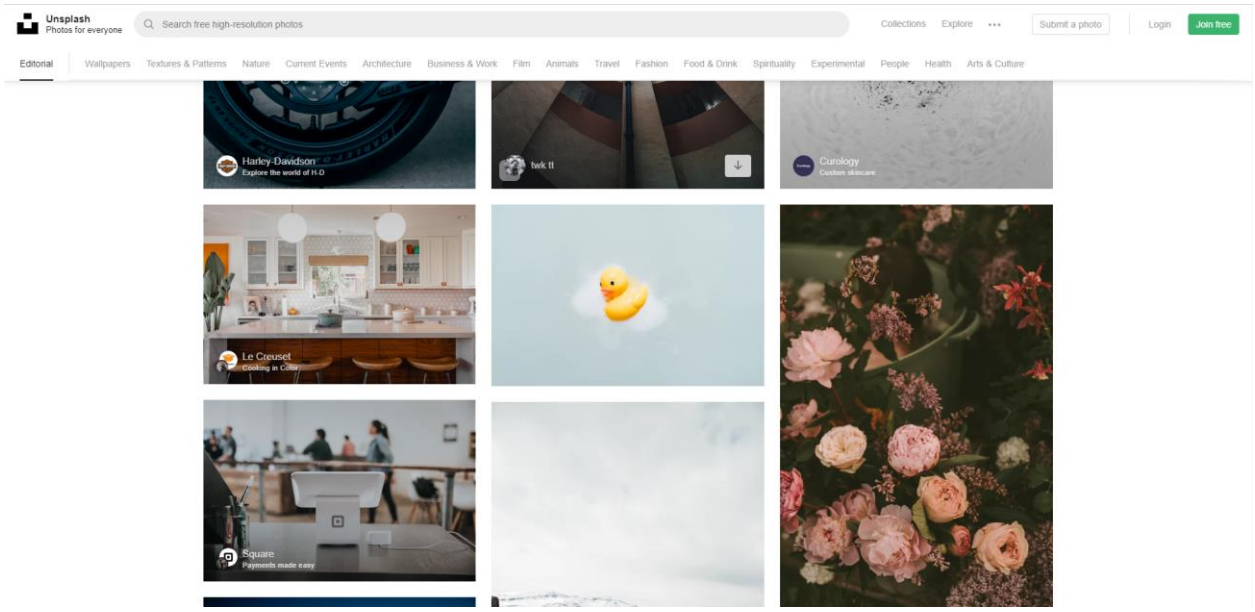

Рисунок 3.2 – Зовнішній вигляд сайту Unsplash

Є, як мінімум, дві чудові причини, чому варто подумати про використання ледачого завантаження зображень на вашому сайті:

- Якщо ваш сайт відображає контент або дає користувачам будь-який функціонал за допомогою JS, то завантаження DOM має вирішальне значення. Звичайно скрипти чекають повного завантаження DOM перед виконанням. На сайті з великою кількістю зображень лінива завантаження (асинхронна завантаження) може стати вирішальним фактором того, чи залишиться користувач або покине сайт.
- Більшість рішень по ледачому завантаженні завантажують зображення тільки, якщо користувач прокрутив сторінку до того місця, де воно потрапляє у view порте. Якщо користувач ніколи не докрутили сторінку до цієї точки, зображення не будуть завантажені. Це значно економить трафік, за що більшість користувачів, особливо користувачі мобільних пристроїв і користувачі на слабких з'єднаннях, скажуть вам спасибі.

3.6.1.1 Просте ліниве завантаження

David Walsh запропонував свій скрипт для ледачої завантаження зображень. Спрощена версія:

Атрибут src в тезі img замінюється на data-src:

```
<img data-src="image.jpg" alt="test image">
```

Рисунок 3.3 – Заміна атрибуту src на data-src

В CSS елементи img з атрибутом data-src приховані. Після завантаження зображення плавно з'являються за допомогою переходів:

```
img {
    opacity: 1;
    transition: opacity 0.3s;
}
img[data-src] {
    opacity: 0;
}
```

Рисунок 3.4 – Плавна поява зображення за допомогою переходів CSS

Далі JS додає атрибут src до всіх img і дає значення відповідного атрибута data-src. Як тільки все зображення завантажилися, скрипт видаляє атрибут data-src з тегів img:

```
[].forEach.call(document.querySelectorAll( selectors: 'img[data-src]'),  argArray: function(img) {
    img.setAttribute('src', img.getAttribute('data-src'));
    img.onload = function() {
        img.removeAttribute('data-src');
    };
});
```

Рисунок 3.5 – Додавання атрибуту src та видалення атрибуту data-src

З іншого боку, цей метод не включає в себе функціонал завантаження по прокручуванні сторінки. Іншими словами, браузер завантажує всі зображення, незалежно від того, прокрутив користувач сторінку чи ні. Сторінка завантажується швидше, так як зображення завантажуються після HTML. Тим не менш, ви не економите трафік.

3.6.1.2 Ліниве завантаження з прогресивним поліпшенням

Robin Osborne запропонував геніальне рішення на основі прогресивного поліпшення. У його методі лінива завантаження на JS вважається поліпшенням для звичайного HTML і CSS.

Чому саме прогресивне поліпшення? Якщо показувати зображення за допомогою JS, що буде, якщо JS відключити, або якщо виникне помилка, яка заблокує виконання скрипта? У такому випадку без прогресивного поліпшення користувачі взагалі не побачать зображень. Не дуже добре.

У цієї техніки ряд переваг:

- Техніка прогресивного поліпшення гарантує користувачам доступ до контенту.
- Метод підходить не тільки під ситуації з недоступним JS, але і під ситуації, коли JS зламаний: бо скрипти схильні до помилок, особливо в середовищі з великою кількістю запущених скриптів.

- Працює лінива завантаження зображень по прокручуванні. Якщо користувач не прокрутить сторінку до потрібного місця, зображення не буде завантажено.
- Метод не спирається на зовнішні залежності, тобто не потрібні фреймворки і плагіни.

3.6.1.3 Плагін jQuery для ледачого завантаження

Швидка і проста альтернатива написання власної ледачою завантаження - JavaScript / jQuery плагін, який зробить всю найскладнішу роботу за вас.

Lazy Load XT - багатофункціональний jQuery плагін. Можна завантажити спрощену версію jquery.lazyloadxt.js, в якій доступна тільки лінива завантаження. Або ж можна використовувати розширену версію плагіна jquery.lazyloadxt.extra.js. У розширеній версії за допомогою ледачою завантаження можна завантажувати iframe, відео і все теги з атрибутом src.

Щоб підключити Lazy Load XT в проект, додайте jQuery-бібліотеку перед закриває тегом з посиланням на один з двох вищезгаданих файлів. наприклад:

```
<script src="jquery.js"></script>
<script src="jquery.lazyloadxt.js"></script>
```

Рисунок 3.6– Підключення JQuery бібліотек

Необхідно замінити в документі, в зображеннях, атрибут src на data-src:

```
<img data-src="lazy.jpg" width="100" height="100" alt="test image">
```

Рисунок 3.7 – Заміна атрибуту src на data-src

Плагін може ініціалізовуватися сам, або ж можна зробити це вручну. Наприклад, для ініціалізації вибірки елементів потрібно вказати:

```
$ (Elements) .lazyLoadXT ();
```

Рисунок 3.8 – Ініціалізація вибірки

Цей плагін додає безліч аддонів. Лише парочка:

- Якщо додати jquery.lazyloadxt.spinner.css, то під час завантаження зображення можна відображати анімований спінер.
- Якщо підключити до проекту animate.min.css і написати цей рядок $ .lazyLoadXT.onload.addClass = 'animated bounceOutLeft'; в JS-файлі, то можна додати всі ефекти для завантаження зображень з Animate.css. bounceOutLeft можна замінити на будь-який інший ефект.

Переваги Lazy Load XT і його аддонів:

- Підтримується CDN, тобто не потрібно завантажувати скрипти Lazy Load XT на свій сервер.
- Широка підтримка в браузерах, в тому числі в IE6-11 і Opera Mini.
- Ледаче завантаження зображень можна використовувати на сторінці, в прокручуємому контейнері, в макетах з вертикальною і горизонтальною прокруткою. Сценаріїв нескінченно багато.
- За допомогою аддонів можна створювати чудові плавні переходи, додати підтримку ретина екранів, ліниво завантажувати фонові зображення і т.д.
- Можна ліниво завантажувати різні типи медіа.

- У документації показано, як протидіяти випадкам з відключеним JS в браузері.
- Не потрібно підключати повну jQuery-бібліотеку, щоб використовувати цей плагін.

3.6.2 Плагін для ледачого завантаження на JavaScript

bLazy - розумний JS-плагін ледачого завантаження. Більш конкретно: «bLazy - легкий скрипт ледачою завантаження (менш 1.2Кб мінімізована і стисла версія). Він дозволяє ліниво завантажувати і обслуговувати кілька зображень для економії трафіку і запитів на сервер. Якщо користувачі не будуть прокручувати всю сторінку, то сайт буде завантажуватися швидше »

Цей невеликий незалежний плагін дозволяє:

- Ліниво завантажувати вставляємі і фонові зображення.
- Завантажувати різні зображення в залежності від розміру екрану і його розширення.
- Ліниво завантажувати все з атрибутом src, наприклад, iframe, HTML5-відео, скрипти, ігри на unity і т.д.
- Ліниво завантажувати зображення в прокручуємо контейнері.
- Підтримувати старі браузери аж до IE7-8.
- Використовувати CDN, щоб не розміщувати плагін на своєму сервері.

Базова реалізація, розмітка:

```
<Img class = "b-lazy" src = "placeholder.gif" data-src = "image.jpg" alt = "test image">
```

Рисунок 3.9 – Базова реалізація, розмітка

Звичайний тег img потрібно змінити наступним чином:

- Додати клас .b-lazy.
- Використовувати плейсхолдер як значення src. Для економії HTTP-запитів також можна використовувати інлайнові прозорі gif з кодуванням base64. Але через це не буде кешування на наступних сторінках, де використовується одне і те ж зображення.
- Атрибут data-src вказує на зображення, яке необхідно ліниво завантажити.

JS: необхідно ввести простий виклик bLazy і налаштуйте об'єкт по карті опцій:

```
var bLazy = new Blazy ({
    // опції
});
```

Рисунок 3.10 – Простий виклик bLazy

3.6.2.1 Ліниве завантаження з ефектом розмитого зображення

Одним з прикладів такого завантаження є платформа соціальної журналістики Medium. Сайт завантажує головне зображення поста. Спершу ви бачите розмите зображення низької якості, поки завантажується версія з високою роздільною здатністю:

Рисунок 3.11 – Розмитий плейсхолдер на сайті Medium

Рисунок 3.12 – Ліниво завантажений плейсхолдер високої якості на сайті Medium

Можна декількома способами ліниво завантажувати зображення з таким цікавим розмитим ефектом. Оптимальний є техніка від Craig Buckler. Плюси його рішення:

- Продуктивність: всього 463 байти CSS і 1.007 байт мінімізованого JS-коду.
- Підтримка ретина екранів.
- Немає залежностей: немає jQuery та інших бібліотек і фреймворків.
- Використовується техніка прогресивного поліпшення для старих браузерів і поламаного JS.

3.7 Покращення користувацького досвіду

Для компаній в сфері електронної комерції, які завантажують сотні зображень продуктів на сторінку, відкладене завантаження може значно поліпшити продуктивність початкового завантаження сторінки.

Проте багато компаній не вибирають відкладене завантаження, тому що вважають, що це суперечить забезпеченню хорошого користувальницького досвіду - плейсхолдери виглядають непривабливо, час завантаження здається досить тривалим і так далі.

Проте при правильному підході використання відкладеного завантаження, можна досягти досить непоганий результат, що в свою чергу дозволяє покращити, як і порядок завантаження, так і те як це завантаження відбудеться, додати до нього певні графічні ефекти за допомогою JavaScript та CSS. Також є можливість підключення бібліотек, при використанні яких не доведеться прописувати весь код заново, а лише додати певні функції та методи до тих елементів над якими необхідно здійснити ту чи іншу операцію.

Це в свою чергу змінить порядок завантаження елементів на сторінці, але при цьому сторінка буде мати не тільки адекватне відображення елементів, а й графічний супровід, який в свою чергу не буде відлякувати користувачів.

3.7.1 Правильний дизайн плейсхолдерів

Плейсхолдер - то, що відображається на сторінці до завантаження зображення. Зазвичай розробники використовують одноколірний плейсхолдер для зображень або одне зображення в якості плейсхолдера для всіх картинок.

Ми використовували подібний плейсхолдер в нашому прикладі - всюди він пофарбований в суцільний світло-сірий колір. Проте можна зробити краще. Розглянемо приклади використання більш вдалих варіантів плейсхолдерів.

3.8 Плейсхолдер домінуючого кольору

Цей метод давно використовується для результатів пошуку зображень в Google і Pinterest.

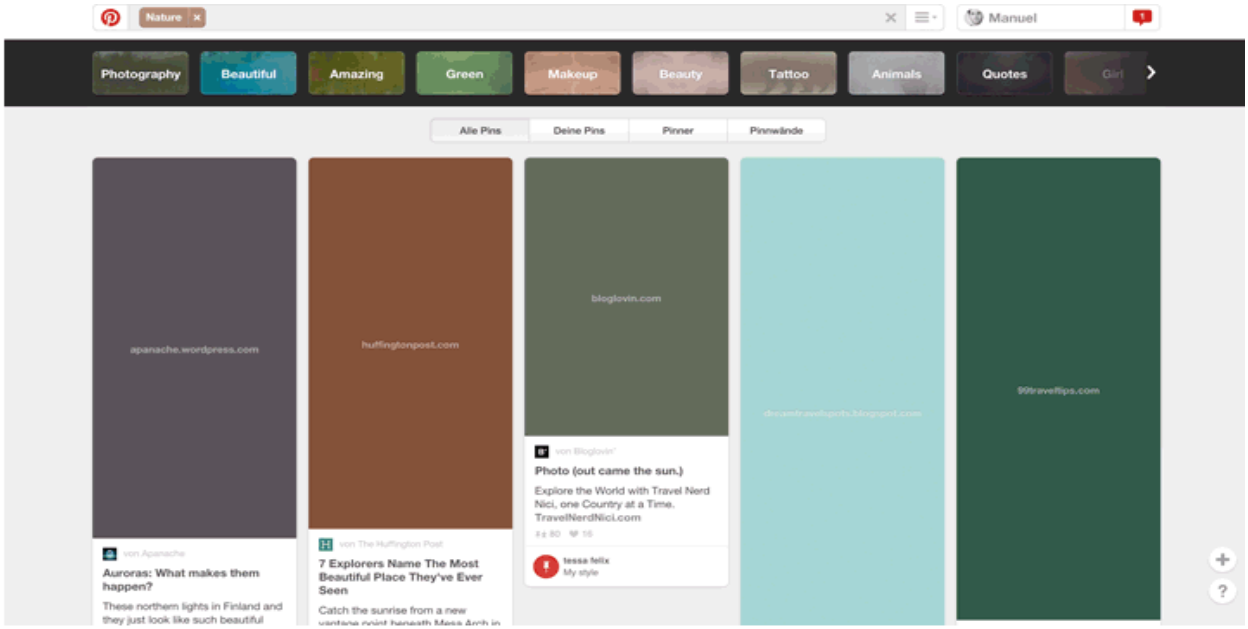

Рисунок 3.13 – Сайт Pintrest на початку формування сторінки

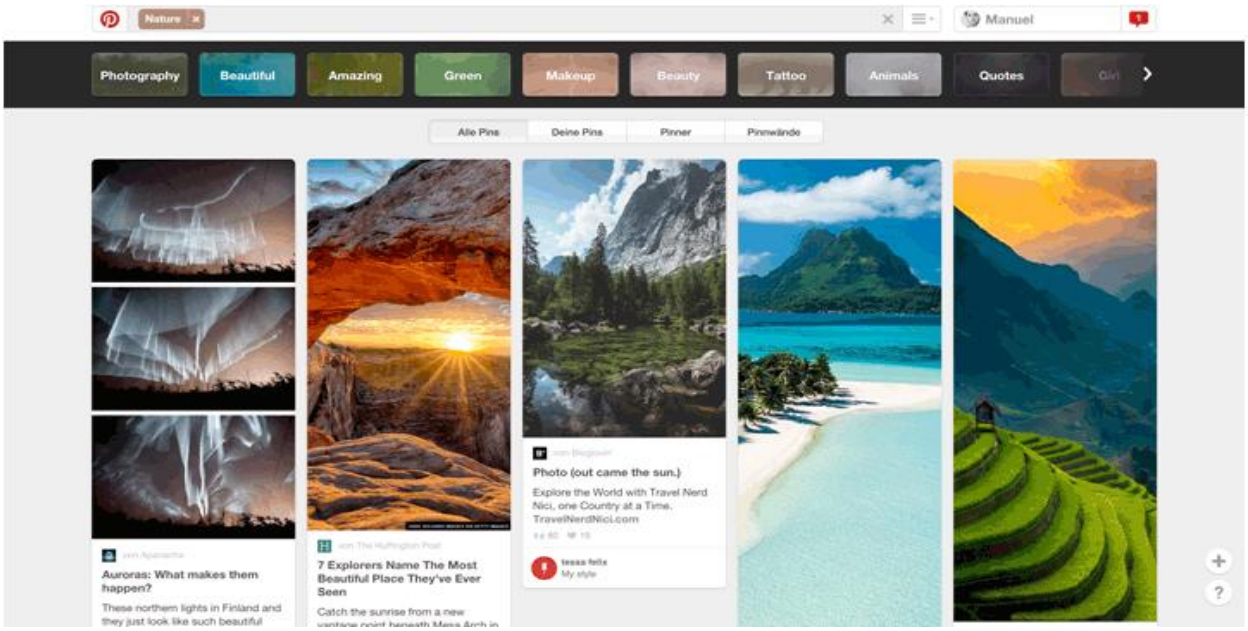

Рисунок 3.14 – Сайт Pintrest після формування сторінки

Може здатися, що це складно реалізувати. Але є простий спосіб - спочатку зменшити зображення до пікселя 1 × 1, а потім масштабувати його до розміру плейсхолдера - грубе наближення, але воно допомагає легко отримати один домінуючий колір. Використовуючи ImageKit, плейсхолдер домінуючого кольору можна отримати за допомогою ланцюгового перетворення, як показано нижче.

```
<!-- Original image at 400x300 -->
<img src="https://ik.imagekit.io/demo/img/image4.jpeg?tr=w-400,h-300" alt="original image" />

<!-- Dominant colour image with same dimensions -->
<img src="https://ik.imagekit.io/demo/img/image4.jpeg?tr=w-1,h-1:w-400,h-300" alt="dominant color
placeholder" />
```

Рисунок 3.15 – Приклад підключення ImageKit для зменшення зображення

Розмір зображення становить всього 620 байт, в порівнянні з вихідним зображенням, яке має розмір 12 500 байт - в 20 разів менше. І це забезпечує більш приємний досвід переходу від плейсхолдера до зображення.

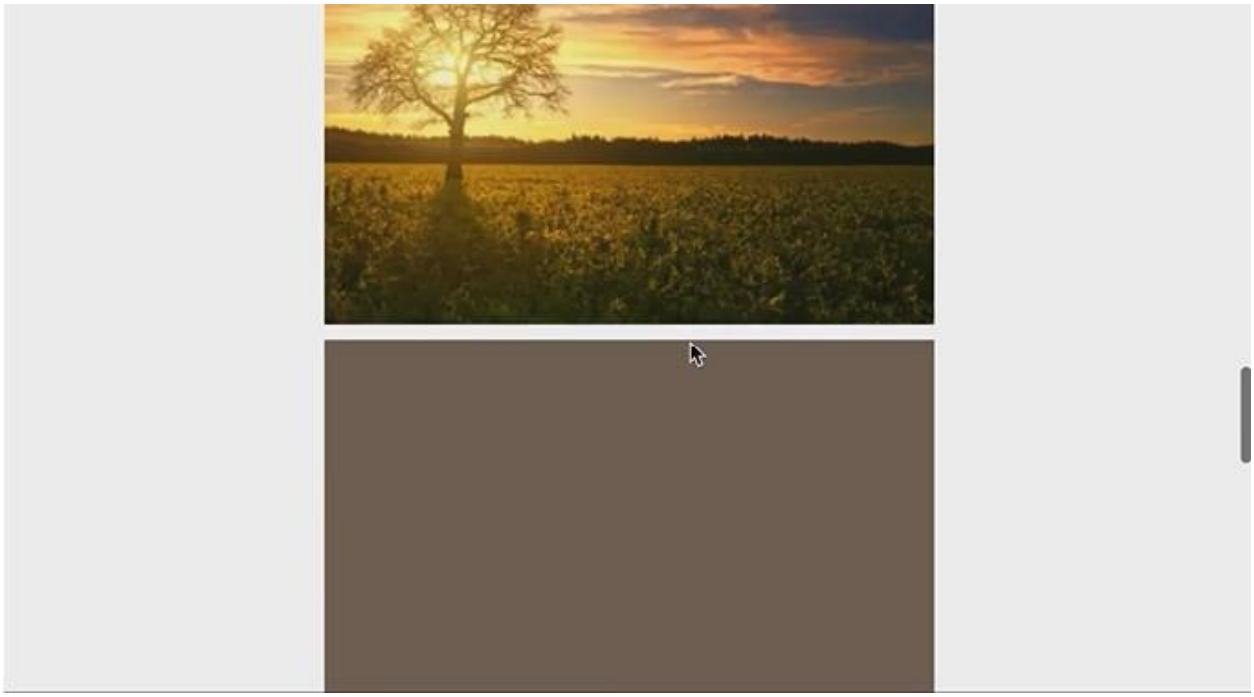

Рисунок 3.16 – Приклад переходу від попереднього зображення до наступного

3.8.1 Плейсхолдер низької якості (LQIP)

Є спосіб розширити наведену вище ідею використання плейсхолдеру домінуючого кольору. Замість одного кольору можна використовувати неякісну розмиту версію вихідного зображення.

Це не тільки виглядає краще, але повідомляє про завантаження і дає користувачеві уявлення про те, чого чекати в реальному зображенні. Цей метод використовують Facebook і Medium для зображень на сайтах і в додатках.

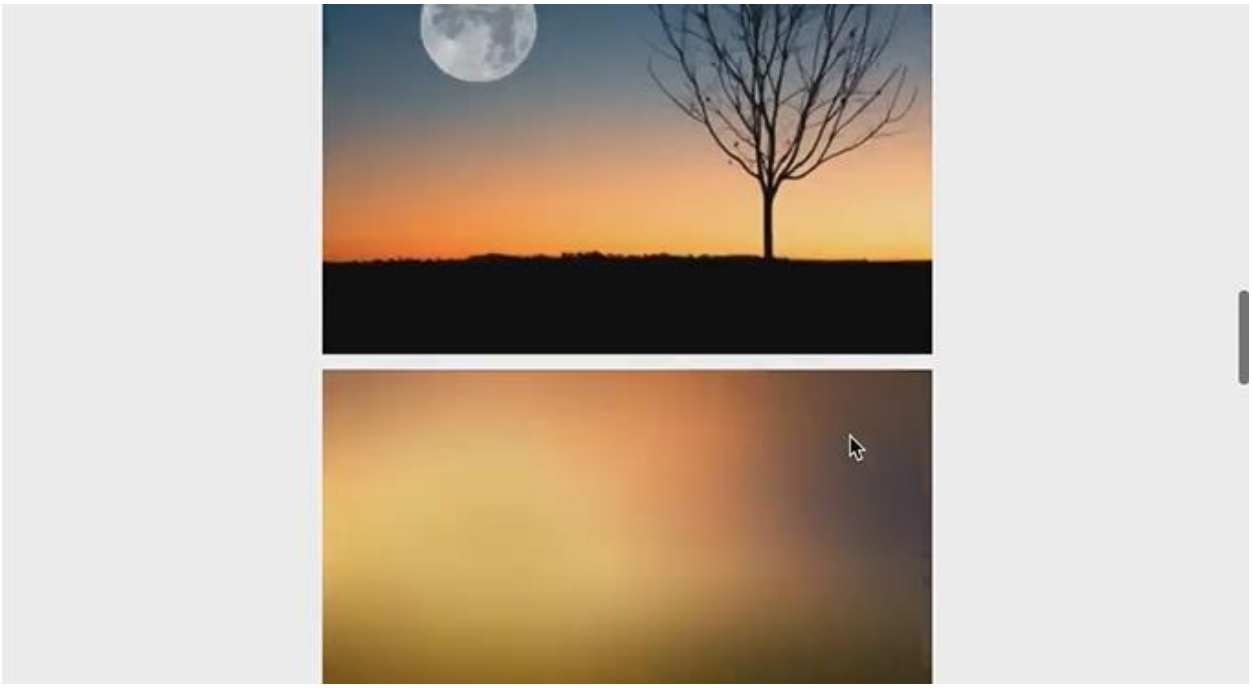

Рисунок 3.17- Приклад переходу від попереднього зображення до наступного з використанням LQIP

Розмір LQIP становить майже в десять разів менше вихідного зображення і значно краще з точки зору користувацького досвіду в порівнянні з будь-яким іншим видом зображень.

3.9 Додавання буферного часу

Часто користувачі швидко прокручують сторінку, і для завантаження і відображення картинки на екрані потрібен якийсь час. Подія «load image» може спрацювати з затримкою, як і плейсхолдери. Це погано впливає на призначений для користувача досвід.

Замість того щоб завантажувати зображення тільки тоді, коли вони точно входять у вікно перегляду, можна завантажити картинки, коли вони, наприклад, в 500 px від краю.

За допомогою Intersection Observer API можна використовувати параметр «` root` »разом з параметром« rootMargin »(працює за стандартним принципом поля CSS), щоб збільшити кордону рамки.

Замість того щоб перевіряти, дорівнює чи є різниця між краєм зображення і краєм вікна нулю, за допомогою методу відстеження подій можна використовувати позитивне число для додавання деякого порогового значення.

У цьому прикладі використовується порогове значення 500 px для завантаження зображень.

Як зрозуміло з відео нижче (уважно стежте за мережевими запитами, що з'являються внизу), при прокручуванні, коли третя зображення знаходиться в полі зору, завантажується п'яте. Коли четвертий зображення з'являється у вікні перегляду, завантажується шосте.

Так ми даємо досить часу для повного завантаження зображень, і в більшості випадків користувач взагалі не побачить плейсхолдер.

3.10 Розробка додатку для браузеру

Розробка буде проводитися бля браузеру Chrome, оскільки даний веб-переглядач має в собі влаштовані засоби для відлагодження програми, перевірки правильності роботи функцій та відловлення помилок на етапі роботи, що значно спрощує знаходження помилок в додаткові, та перевірки на виконання елементів додатку.

3.11 Створення проекту

Перше, що потрібно зробити, це створити проект і всі файли, які потрібні для нашого розширення. Необхідно почати з створення нового каталогу, який зазвичай називається так як і проект, що планується розробитися. У цей каталог

поміщаються всі файли, які потрібні для розширення. Chrome дозволяє завантажити плагін, вказавши директорію, що містить файли з розширенням.

Всі розширення Chrome вимагають наявності файлу маніфесту. Файл маніфесту повідомляє браузеру все, що потрібно для завантаження розширення. Файл manifest.json створюється в директорії проекту.

Далі необхідно завантажити іконку для нашого розширення, це може бути іконка з вільного ресурсу, де не порушуються авторські права, іконка зроблена під замовлення, на яку розробник буде мати повні права, чи намальована власноруч іконка. Це повинен бути PNG-файл 19 × 19px.

Далі створюється HTML-сторінка, яка буде відображатися при натисканні на іконку на панелі додатків в браузері. Для цього додються файли popup.html і popup.js в директорію проекту.

Через обмеження безпеки, не рекомендується використовувати в розширенні JavaScript-код, вбудований в HTML, тому необхідно посилатися на зовнішній файл, що міститиме в собі логіку роботи.

3.12 Створення файлу маніфесту

Тепер, після створення базової структури проекту, потрібно додати опис розширення в файл маніфесту.

Зазвичай в файл manifest.json додається наступний код(Додаток А.1)

Більшість полів в цьому JSON-файлі не потребують додаткового пояснення, зазвичай в ньому прописується: назва, опис, версія додатку, а також іконка для панелі додатків, HTML файл з вікном ,на якому розміщається графічний інтерфейс. Основним розділом є browser_action, де ми визначається, яка іконка буде використовуватися і яка HTML-сторінка повинна відображатися при натисканні на кнопку.

3.13 Створення інтерфейсу

Наступним кроком буде створення інтерфейсу, який буде відображатися при натисканні на іконку.

Інтерфейс не вимагає від себе складних рішень, тому він простий, вікно містить назву програми і кнопку, по якій користувач зможе виконати дії на на всіх чи на поточній сторінці.

У HTML-файлі підключається скрипт. В цьому скрипті буде реалізована логіка нашого розширення, яка буде виконуватися при натисканні на кнопку.

3.14 Реалізація логіки

Остання річ, яку необхідно додати для роботи додатку, це реалізація логіки, яка повинна виконуватися при натисканні на кнопку.

Також реалізовані два різні методи реалізації логіки: метод домінуючого кольору та зображення низької якості. Вибрати метод можна завдяки списку, що розгортається та підтвердити вибране натиском на кнопку.

Це в свою чергу викличе обробник подій, що знаходиться в скрипті.

ВИСНОВКИ

В даній дипломній роботі було розглянуто методи оптимізації інформаційного поля веб-сторінки, які в свою чергу являють собою різні підходи до оптимізації. Основним методом для втілення було взято реалізацію через додаток для браузеру. Оскільки даний метод дозволяє працювати на любій операційній системі, де встановлено веб-браузер, а також використання можливе в різних браузерах, що в свою чергу спрощує користувачеві моменти, пов'язані з використанням веб-браузеру.

В першому розділі розглянуто та дано характеристику інших аналогічних рішень, що дозволяють оптимізувати представлення та швидкість завантаження веб-сторінок, таких як Турбо режими в браузерах Opera та Яндекс, додатку для браузеру Data saver, що в свою чергу працює по схожій схемі з режимом Турбо в вищезазначених браузерах, а також представлено браузер Vivaldi, що має в собі доволі сильну базу для оптимізації, але не раціональний, бо користувачеві доведеться переносити свої дані в новий браузер та звикати до інтерфейсу. Також вибрано основним способом реалізації додаток для браузеру. Дано загальну характеристику використання додатків та розглянуто основні моделі їх використання.

В другому розділі розглянуто основний метод формування веб-сторінки, чому і на які моменти потрібно звертати увагу при розробці та підтримці сайту. На що потрібно звернути увагу для забезпечення максимальної швидкості завантаження веб-сторінки та її вимірюванні? Чому потрібно вибирати оптимальні серверні ресурси та оптимізувати їх програмну частину? Також необхідно оптимізувати СУБД для швидкого відклику та отримання даних з неї. В клієнтській частині розглянуто оптимізацію використання скриптів JavaScript та CSS, які шрифти бажано використовувати та їх застосування для максимальної оптимізації. Наведені методи стиснення інформації, використання яких значно полегшує завантаження, видаючи стиснену версію інформації, якщо така є в

наявності. Порівняно розміри зображень з форматами WebP, JPG, GIF, PNG та виявлено, що самим оптимальним для швидкого завантаження є WebP. Розглянуто метод CDN для розвантаження мережі при великій кількості відвідувачів ресурсів.

В третьому розділі описано розробку програмного продукту, додатку для браузеру. Додатки бля браузеру є оптимальним доповненням до сайтів, по причині, що можна розширити функціонал сайту не змінюючи основного коду. Передбачити масу сценаріїв використання певного ресурсу неможливо, бо через це оптимізація буде викликати проблеми, які проблемно буде вирішити. Завдяки додатку, кожен користувач чи розробник може знайти для себе ту модель використання додатку, яку він захоче. В даному проекті це зменшення навантаження за рахунок зображень, оскільки вони є доволі великим та об'ємним носієм. Основна ідея відкласти завантаження того зображення, що користувачеві не потрібно в даний момент. Це можна назвати лінивим завантаженням, оскільки поки інформація не потрібна, вона не буде завантажена. Використано методи «зображення домінуючого кольору» та «зображення низької якості», що дозволяє зберегти привабливий зовнішній вигляд та оптимальну оптимізацію.

Можна зробити висновок, що даний метод є молодим та перспективним для веб-сторінок, оскільки допомагає оптимізувати відображення та завантаження веб-сторінок. А використання його через доповнення для браузеру спрощує використання за рахунок можливості працювати в різних браузерах та на різних операційних системах. Хоча підхід, що використовує даний додаток, має й свої недоліки, основним з яких є те, що сайти розробляються різними технологіями з використанням фреймворків reactjs, angularjs, vuejs та інші. Це може викликати певні проблеми у виявленні зображень, для оптимізації веб-сторінки.

ПЕРЕЛІК ПОСИЛАНЬ

1. Web Font Optimization - Режим доступу: https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/webfont-optimization?hl=ru/. – Дата доступу: 20.04.2019.
2. Что такое плагин – Режим доступу: https://ktonanovenkogo.ru/voprosy-i-otvety/chto-takoe-plaginy-skachat-kak-ustanovit-obnovit.html/. – Дата доступу: 15.04.2019.
3. Составление DNS запроса – Режим доступу: https://habr.com/ru/post/346098/. – Дата доступу: 17.04.2019.
4. Five Techniques to Lazy Load Images for Website Performance / Maria Antonietta Perna// - Режим доступу : https://www.sitepoint.com/five-techniques-lazy-load-images-website-performance/. – Дата доступу: 01.05.2019.
5. Dominant Colors for Lazy-Loading Images – Режим доступу: https://manu.ninja/dominant-colors-for-lazy-loading-images/. – Дата доступу: 05.05.2019
6. Lazy Loading Images – The Complete Guide / Rahul Nanvani // - Режим доступу: https://imagekit.io/blog/lazy-loading-images-complete-guide/. – Дата доступу:07.05.2019.
7. Расширения браузера, их виды и польза / Evergreen // - Режим доступу: https://evergreens.com.ua/ru/articles/browser-extensions.html/. – Дата доступу 15.04.2019.
8. Content Scripts – Режим доступу: https://developer.chrome.com/extensions/content_scripts/. – Дата доступу: 21.042019.
9. Экономия трафика — расширение Data Saver – Режим доступу: https://vellisa.ru/extension-data-saver/. – Дата доступу :12.04.2019.

10. Создание расширения – Режим доступу: https://developer.mozilla.org/ru/docs/Building_an_Extension/. – Дата доступу: 17.05.2019.
11. Create a Chrome extension to modify a website's HTML or CSS – режим доступу: https://blog.lateral.io/2016/04/create-chrome-extension-modify-websites-html-css/. – Дата доступу: 18.05.2019.
12. Николай Лавинский. Как ускорить загрузку сайта - Режим доступу: https://habr.com/ru/company/netologyru/blog/337842/. – Дата доступу: 15.04.2019.
13. What is SSL, TLS and HTTPS? – режим доступу: https://www.websecurity.symantec.com/security-topics/what-is-ssl-tls-https/. – Дата доступу: 3.05.2019.
14. DIFFERENCE BETWEEN HTTP, HTTPS, SSL, TLS? – Режим доступу: https://www.idiotinside.com/2018/02/07/http-https-ssl-tls/. – Дата доступу: 3.05.2019.
15. Методы сжатия данных в Сети: zopfli, brotli, sdch – Режим доступу: https://www.searchengines.ru/metody-szhatiya-dannyh-v-seti-zopfli-brotli-sdch.html/. – Дата доступу: 29.04.2019.

ДОДАТОК А. ЛІСТИНГ РОЗРОБЛЕНОЇ ПРОГРАМИ

“manifest.json”

```
{
  "manifest_version": 2,
  "name": "WEB page optimization",
  "description": "This extension will optimize images on web pages",
  "version": "1.0",
  "browser_action": {
    "default_icon": "icon.png",
    "default_popup": "popup.html"
  },
  "content_scripts": [
    {
      "matches": [ "http://extension.target.url/" ],
      "js": [ "jquery.js", "background.js" ],
      "run_at": "document_end"
    }
  ],
  "web_accessible_resources": [
    "/script.js"
  ]
}
```

“popup.html”

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Page optimizator</title>
    <link src="style.css"/>
    <script src="background.js"></script>
    <script type="text/javascript" src="jquery.js"></script>
</head>
<body>

<h1>Optimization plugin</h1>
<select>
    <option disabled>Виберіть метод</option>
    <option id="Color">По Кольору</option>
    <option id="Blurred">Розмиття</option>
</select>

<input type="submit" value="Старт">
</body>
</html>
```

“background.js”

```
jQuery(document).ready(function($){
        $.get(chrome.extension.getURL('/script.js'),
            function(data) {
                var script = document.createElement("script");
                script.setAttribute("type", "text/javascript");
                script.innerHTML = data;
                document.getElementsByTagName("head")[0].appendChild(script);
                document.getElementsByTagName("body")[0].setAttribute("onLoad",
"injected_main();");
            }
        );
});
```

“script.js”

```
function injected_main() {
    [].forEach.call(document.querySelectorAll('img[src]'), function (img) {
        img.setAttribute('data-src', img.getAttribute('src'));
        img.onload = function () {
            img.removeAttribute('src');
        };
    });
    if (document.getElementById("Color")) {
```

```
document.addEventListener("DOMContentLoaded", function () {
    let lazyImage;
    if ("IntersectionObserver" in window) {
        lazyImage = document.querySelectorAll(".lazy");
        let lazyObserver = new IntersectionObserver(function (entries, observer) {
            entries.forEach(function (entry) {
                if (entry.isIntersecting) {
                    let img = entry.target;
                    img.src = img.dataset.src;
                    img.classList.remove("lazy");
                    lazyObserver.unobserve(img);
                }
            });
        });
        lazyImage.forEach(function (img) {
            lazyObserver.observe(img);
        });
    } else {
        let lazyloadTimeout;
        lazyImage = document.querySelectorAll(".lazy");
        function lazyload() {
            if (lazyloadTimeout) {
```

```
            clearTimeout(lazyloadTimeout);
        }
        lazyloadTimeout = setTimeout(function () {
            let scrollUp = window.pageYOffset;
            lazyImage.forEach(function (img) {
                if (img.offsetTop < (window.innerHeight + scrollUp)) {
                    img.src = img.dataset.src;
                    img.classList.remove('lazy');
                }
            });
            if (lazyImage.length == 0) {
                document.removeEventListener("scroll", lazyload);
                window.removeEventListener("resize", lazyload);
                window.removeEventListener("orientationChange", lazyload);
            }
        }, 20);
    }
    document.addEventListener("scroll", lazyload);
    window.addEventListener("resize", lazyload);
    window.addEventListener("orientationChange", lazyload);
  }
})
```

```
if (document.getElementById("Blurred")) {
    document.ready(function () {
        let lazyImage;
        if ("IntersectionObserver" in window) {
            lazyImage = document.querySelectorAll(".lazy");
            let lazyObserver = new IntersectionObserver(function (entries, observer) {
                entries.forEach(function (entry) {
                    if (entry.isIntersecting) {
                        let img = entry.target;
                        img.src = img.dataset.src;
                        img.classList.remove("lazy");
                        lazyObserver.unobserve(img);
                    }
                });
            });
            lazyImage.forEach(function (img) {
                lazyObserver.observe(img);
            });
        } else {
            let lazyloadTimeout;
            lazyImage = $(".lazy");
```

```
function lazyload() {
    if (lazyloadTimeout) {
        clearTimeout(lazyloadTimeout);
    }
    lazyloadTimeout = setTimeout(function () {
        let scrollUp = $(window).scrollUp();
        lazyImage.each(function () {
            let el = $(this);
            if (el.offset().top - scrollUp < window.innerHeight) {
                let url = el.attr("data-src");
                el.attr("src", url);
                el.removeClass("lazy");
                lazyImage = $(".lazy");
            }
        });
        if (lazyImage.length == 0) {
            $(document).off("scroll");
            $(window).off("resize");
        }
    }, 20);
}
```

```
                $(document).on("scroll", lazyload);
                $(window).on("resize", lazyload);
            }
        })
    }
}
```

“style.css”

```
html, body
{
    height: 100vh;
}
#Header
{
    width: 960px;
    height: 150px;
}
#Content
{
    height: 100%;
    width: 960px;
}
```